University of Alberta

Data Analysis and Prediction Models in Oil Sands Mining

by

Alexander Medin

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements for the degree of Masters of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall 2003

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisisitons et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-87834-1
Our file Notre référence
ISBN: 0-612-87834-1

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

Canada

University of Alberta

Library Release Form

Name of Author: Alexander Medin

Title of Thesis: Data Analysis and Prediction Models in Oil Sands Mining

Degree: Masters of Science

Year this Degree Granted: 2003

Permission is hereby granted to the University of Alberta to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatsoever without the author's prior written permission.

June 2, 2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Data Analysis and Prediction Models in Oil Sands Mining submitted by Alexander Medin in partial fulfilment of the requirements for the degree of Masters of Science.

Dr. Witold Pedrycz

Dr. Jozef Szymanski

Dr.Samuel Frimpong

Dr. Marek Reformat

Date: May 30, 2005

University of Alberta

Abstract

Proper operation and maintenance of equipment are essential to effective mining operations in oil sands industry relying heavily on truck and shovel methods. The operational goal is to minimize truck damage while maximizing production. To effectively implement this understanding, continuous monitoring and predictive modeling of truck shovel parameters is needed. Of paramount importance is to predict heavy truck drivers' exposure to vibration during regular tar sand hauling operation.

This research focuses on analysis and modeling of experimental datasets collected from tar sand heavy hauler operations in summer conditions. The stress-related accelerations were measured by tri-axial accelerometers mounted on the seat pan of the truck driver. Several predictive models were developed to investigate the effect of important input parameters of a hauling truck (suspension cylinders pressure, ground speed, payload weight, rack and pitch) on these accelerations magnitudes. Best performing model was found experimentally, able to predict acceleration changes with reasonable accuracy.

Acknowledgements

I would like to express my recognition and gratitude to Dr. Witold Pedrycz for his supervision of my research and for many valuable ideas that made a significant impact on this thesis.

I am thankful to Dr. Jozef Szymanski for the data he provided for this research, and for the explanations of the nature of the data and their collection process. I acknowledge him, as well as Dr. Samuel Frimpong of the University of Alberta School of Mining, and NSERC for financial support of this research.

Special thanks to Shauna Rae for proof-reading of the thesis, and to other graduate students with the Department of Electrical and Computer Engineering for their helpful ideas, answers for some specific questions, and, overall, for their cooperation during our study at the University of Alberta.

Finally, thanks to my family for their patience and moral support throughout my graduate program.

Table of contents

Chapter 1. Introduction 1
1.1. Data analysis: state of the art 1
1.2. The problem to be considered 2
1.3. Objectives of the thesis 3
1.4. Approach and methodology 4
1.5. Literature review 5
Chapter 2. Description of the data analyzed 8
2.1. Industry background 8
2.2. Methodology and raw data collection 10
2.3. Importance of data preprocessing 12
2.4. Creating a working dataset from the original raw data 13
2.5. Statistical information about the working dataset 22
2.6. Possible models of data analysis and prediction 25
2.7. Conclusions 27
Chapter 3. Artificial neural networks 28
3.1. Neural Networks Foundations 29
3.1.1. Neurobiological analogy 29
3.1.2. Principles of learning process 30
3.1.3. Back-propagation learning algorithm 32
3.1.4. Gradient descent principle in minimizing the error 33
3.1.5. Supervised versus non-supervised learning 35
3.1.6. Problem of neural network over-fitting 36
3.1.7. Importance of data preprocessing in neural networks 38
3.2. Theoretical base of the model 40
3.2.1. Feed-forward neural networks with one hidden layer 40
3.2.1.1. Topology of a neural network with one hidden layer 40
3.2.1.2. Number of neurons in hidden layer 42
3.2.1.3. Types of transfer functions 43
3.2.1.4. Implementation of the gradient descent algorithm 45
3.2.1.5. Influence of learning rate 48
3.2.1.6. Momentum in learning 48
3.2.2. Neural network with several hidden layers 49
3.2.2.1. Topology of multi-layer network 49
3.2.2.2. Computing partial derivatives 51
3.2.3. Neural network with feedback from the target outputs 52
3.2.3.1. Rationale behind introducing a feedback 52
3.2.3.2. Method of feedback implementation 54
3.3. Feed-forward neural network model applied to the analyzed dataset 55
3.3.1. Experimenting with number of learning epochs 56
3.3.2. Experimenting with learning rate and with number of hidden neurons in a network with one hidden layer 58
3.3.3. Cross-validation (repeating the experiments for 10 different data partitions) 65
3.3.4. Comparing neural networks with different transfer functions 68
3.3.5. Introducing the momentum into the learning process 69
3.3.6. Analyzing the effect from introducing a second hidden layer 69
3.3.7. Introducing feedback from the target outputs 70
3.3.8. Experimenting with multi-cycle feedback 79
3.4. Experimenting with feedback separately on different outputs 83
3.5. Conclusions 84
Chapter 4. Local models based on data clustering 86
4.1. Concept of clusters as receptive fields 87
4.2. Fuzzy C-Means clustering algorithm 91

4.3. Data visualization as a pre-processing tool before building a model 93
4.4. Local linear models 106
4.4.1. Least square solution for a linear regression 106
4.4.2. Theoretical base of the local linear regressions model 108
4.4.3. Applying the model to the analyzed dataset 111
4.4.4. Introducing a threshold at data allocation 116
4.5. Takagi – Sugeno model and its application to the analyzed data 122
4.5.1. Theoretical base of the model 122
4.5.2. Applying the model to the analyzed dataset 126
4.5.3. Analysis of results 133
4.6. Conclusions 134
Chapter 5. Radial basis function neural networks 135
5.1. Theoretical base of the model 135
5.1.1. Concept overview 135
5.1.2. Topology of the RBF network 136
5.1.3. Mathematical description of RBF network operation 138
5.1.4. Training the RBF network 140
5.1.5. Computing the weights through least square solution 143
5.2. Applying the model to the analyzed dataset 145
5.2.1. Experimenting with the number of clusters and FCM termination criteria 145
5.2.2. Experimenting with types of radial basis functions 148
5.2.3. Computing RBF as partition matrix values 150
5.2.4. Experimenting separately on different outputs 152
5.2.5. Experimenting with dynamic RBF network models 152
5.2.6. Computing weights through iterative learning starting from random values 153
5.3. Conclusions 157
Chapter 6. General conclusion 158

List of tables

Table 2.1 Fragment of the first spreadsheet of the raw data workbook 103num01cr. 14
Table 2.2 Fragment of the second spreadsheet of the raw data workbook 103num01cr. 14
Table 2.3 Data measurements interpreted as inputs 15
Table 2.4 Data measurements interpreted as outputs 15
Table 2.5 Minimum, mean, maximum values and standard deviation for each of the data inputs and outputs throughout the entire dataset (3170 data points) 22
Table 2.6 Correlation matrix between inputs and outputs of the working dataset 22
Table 3.1 Performance index RMSE for each experiment with static neural network with one hidden layer, all sigmoid transfer functions, no momentum, no feedback. 59
Table 3.2 Performance index RMSE for each experiment with two hidden layers. 70
Table 3.3 Performance index RMSE for each experiment with one-cycle feedback. 71
Table 3.4 Performance index RMSE for each experiment with two-cycle feedback. 79
Table 3.5 Weights of connections between layers for the best performing neural network model. 85
Table 4.1 Effect of clustering on standard deviation of the outputs 99
Table 4.2 Prototypes of 5 clusters after performing FCM algorithm 100
Table 4.3 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) 113
Table 4.4 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster). 114
Table 4.5 Average cross-validation results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix). 115
Table 4.6 Average cross-validation results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster). 115
Table 4.7 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.4 117
Table 4.8 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.4 117
Table 4.9 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.5 117
Table 4.10 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.5 117
Table 4.11 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.6 118
Table 4.12 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.6 118
Table 4.13 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.7 118
Table 4.14 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.7 119
Table 4.15 Takagi – Sugeno model performance with only input variables used for clustering and FCM termination criterion of 0.01. 128
Table 4.16 Takagi – Sugeno model performance with both inputs and outputs used for clustering and FCM termination criterion of 0.01. 128
Table 4.17 Takagi – Sugeno model performance with only input variables used for clustering and FCM termination criterion of 0.00001. 128
Table 4.18 Takagi – Sugeno model performance with both inputs and outputs used for clustering and FCM termination criterion of 0.00001. 128
Table 5.1 Performance of RBF networks with different number of clusters 145
Table 5.2 Performance of RBF network for the original dataset and average performance for 10 different partitions into training and testing parts (original sequence and 9 randomly shuffled). 147
Table 5.3 Performance of models with different types of radial basis functions 149

Table 5.4 Performance of RBF network for the original dataset and average performance for 10 different partitions into training and testing parts (original sequence and 9 randomly shuffled). 151
Table 6.1 Comparative analysis of performance of all types of the models 158

List of figures

Figure 2.1 The Caterpillar 797 haul truck 9
Figure 2.2 Suspension cylinder to frame attachment locations 11
Figure 2.3 Fragment of the raw dataset representing increase of the truck payload measured during the beginning of one of the working cycles 17
Figure 2.4 Inputs 1 to 4 of the working dataset: ground speed, machine pitch, machine rack, and payload 18
Figure 2.5 Inputs 5 to 8 of the working dataset: pressure in each of the 4 suspension cylinders .. 19
Figure 2.6 Outputs 1 to 3 of the working dataset: X, Y, Z components of the floor acceleration . 20
Figure 2.7 Outputs 4 to 6 of the working dataset: X, Y, Z components of the seat acceleration... 21
Figure 2.8 Histograms of the inputs 24
Figure 2.9 Histograms of the outputs 24
Figure 2.10 Model built to predict the acceleration parameters from truck and shovel parameters 26
Figure 3.1 Sigmoid transfer function 39
Figure 3.2 First derivative of the sigmoid function 39
Figure 3.3 Topology of a feed-forward neural network with one hidden layer 41
Figure 3.4 Plot of arctangent function and its first derivative 44
Figure 3.5 Plot of hyperbolic tangent function and its first derivative 44
Figure 3.6 Topology of a feed-forward neural network with two hidden layers 50
Figure 3.7 Topology of a neural network with feedback from the target outputs 53
Figure 3.8 Tracing the performance index during 1000 training epochs 57
Figure 3.9 Tracing the performance index during 4000 training epochs 57
Figure 3.10 Output y1 (Floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training dataset, lower plot – fragment of testing dataset. 60
Figure 3.11 Output y2 (Floor Y acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training dataset, lower plot – fragment of testing dataset. 61
Figure 3.12 Output y3 (Floor Z acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training dataset, lower plot – fragment of testing dataset. 62
Figure 3.13 RMSE as a function of number of neurons in the hidden layer and learning rate. Upper plot for training dataset, lower plot – for testing dataset. Output layer sigmoid. 64
Figure 3.14 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for cross-validation experiments 66
Figure 3.15 Testing RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for cross-validation experiments 67
Figure 3.16 RMSE for feedback experiments as a function of number of hidden neurons and learning rate. Upper plot for training dataset, lower plot – for testing dataset. 72
Figure 3.17 Floor X acceleration: dotted line – target, solid line – model output. Upper plot represents fragment of training data, lower plot – fragment of testing data. One-cycle feedback. 73
Figure 3.18 Floor Y acceleration: dotted line – target, solid line – model output. Upper plot represents fragment of training data, lower plot – fragment of testing data. One-cycle feedback. 74
Figure 3.19 Floor Z acceleration: dotted line – target, solid line – model output. Upper plot represents fragment of training data, lower plot – fragment of testing data. One-cycle feedback. 75
Figure 3.20 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for one-cycle feedback cross-validation experiments. 77

Figure 3.21 Testing RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for one-cycle feedback cross-validation experiments. 78
Figure 3.22 Convergence of the training performance index for the model with two-cycle feedback. 80
Figure 3.23 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for two-cycle feedback cross-validation experiments. 81
Figure 3.24 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for two-cycle feedback cross-validation experiments. 82
Figure 4.1 Local models as receptive fields to process the data 90
Figure 4.2 Scatter plot for the pair of inputs x2 (machine pitch) vs. x1 (ground speed) 94
Figure 4.3 Scatter plot for the pair of inputs x3 (machine rack) vs. x1 (ground speed) 94
Figure 4.4 Scatter plot for the pair of inputs x4 (payload) vs. x1 (ground speed) 95
Figure 4.5 Scatter plot for the pair of inputs x3 (machine rack) vs. x2 (machine pitch) 95
Figure 4.6 Scatter plot for the pair of inputs x4 (payload) vs. x2 (machine pitch) 96
Figure 4.7 Scatter plot for the pair of inputs x4 (payload) vs. x3 (machine rack) 96
Figure 4.8 Scatter plot for the pair of output y1 (floor X acceleration) vs. input x1 (ground speed) 97
Figure 4.9 Scatter plot for the pair of output y2 (floor Y acceleration) vs. input x2 (machine pitch) 97
Figure 4.10 Scatter plot for the pair of output y3 (floor Z acceleration) vs. input x3 (machine rack) 98
Figure 4.11 Scatter plot for the pair of output y6 (seat Z acceleration) vs. input x4 (payload) 98
Figure 4.12 Output y1 (Floor X acceleration) values sorted by 5 clusters 102
Figure 4.13 Output y2 (Floor Y acceleration) values sorted by 5 clusters 102
Figure 4.14 Output y3 (Floor Z acceleration) values sorted by 5 clusters 103
Figure 4.15 Output y4 (Seat X acceleration) values sorted by 5 clusters 103
Figure 4.16 Output y5 (Seat Y acceleration) values sorted by 5 clusters 104
Figure 4.17 Output y6 (Seat Z acceleration) values sorted by 5 clusters 104
Figure 4.18 Plot of two first components after PCA performed on 8 inputs of the dataset 105
Figure 4.19 Plot of two first components after PCA performed on all columns (8 inputs and 6 outputs) of the dataset 105
Figure 4.20 Output y1 (Floor X acceleration): dotted line – target, solid line – model output. Upper plot represents cluster 1 of the training dataset, lower plot – cluster 1 of the testing dataset. 120
Figure 4.21 Output y3 (Floor Z acceleration): dotted line – target, solid line – model output. Upper plot represents cluster 1 of the training dataset, lower plot – cluster 1 of the testing dataset. 121
Figure 4.22 TS model, output y1 (floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data. 130
Figure 4.23 TS model, output y2 (floor Y acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data. 131
Figure 4.24 TS model, output y3 (floor Z acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data. 132
Figure 5.1 Topology of the RBF network with n inputs, c basis functions, and m outputs 137
Figure 5.2 The Gaussian radial basis function with centre $v = 0$ and radius $\sigma = 1$ 139
Figure 5.3 RBF network performance index RMSE as a function of number of clusters in data partition: dotted line – training RMSE, solid line – testing RMSE. Type of RBF – Gaussian. 146

Figure 5.4 Average (for 10 data partitions) performance index RMSE as a function of number of clusters: dotted line – training RMSE, solid line – testing RMSE. Type of RBF – Gaussian. 147
Figure 5.5 RBF network performance index RMSE as a function of number of clusters: dotted line – training RMSE, solid line – testing RMSE. Type of RBF – inverse multiquadric.... 150
Figure 5.6 RBF network performance index RMSE as a function of number of clusters: dotted line – training RMSE, solid line – testing RMSE. Values of RBF are taken from partition matrix. 151
Figure 5.7 RBF model, output y1 (floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data. 154
Figure 5.8 RBF model, output y2 (floor Y acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data. 155
Figure 5.9 RBF model, output y3 (floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data. 156
Figure 6.1 Comparison of training and testing performance of all types of the models 159

List of abbreviations

RBF	Radial Basis Function
FCM	Fuzzy C-Means
VIMS	Vital Information Management System
PCA	Principal Component Analysis
MSE	Mean Squared Error
RMSE	Root Mean Squared Error

Chapter 1. Introduction

1.1. Data analysis: state of the art

Increasing amounts of data being constantly generated and accumulated by more and more businesses and organizations created an urgent and ever growing need in understanding and practical usage of those huge data repositories. The widening gap between data generation and the ability to make sense of these data led to appearance of different methods of data analysis including such categories as rough sets, fuzzy sets, Bayesian methods, evolutionary computing, machine learning, neuro-computing, clustering, and preprocessing. These and other methods of data analysis are used in a process called knowledge discovery, to reveal new pieces of knowledge from large databases [3].

There exists an extensive history of research that applies the data analysis theories and methods to the data collected by industry, in attempts to reveal their internal structure, to find interesting interrelationships, to derive rigorous rules explaining the data using mathematical foundations, and also to predict the system behavior on the basis of observed regularities. The purpose of data analysis is to generate relevant information about the vast amounts of data that a company or an organization is able to collect during its industrial operations or business processes. This knowledge may be crucial for better understanding the operations and processes, ensuring safety for the people involved, minimizing incurred costs, more reasonable use of equipment, more precise planning, and adequate decision making.

1.2. The problem to be considered

The data for this research was provided by an oil sands mining company that operates an open pit mine in northern Alberta, Canada. As in any other business, the company's objectives are cost efficient mining operations, allowing it to maximize production while minimizing operational expenses and risks. After recent modifications of the operational processes used by the company, as well as by the industry in general, in present day mining operations, the company runs the haul trucks equipped with sophisticated monitoring systems. This equipment allows the operator to record in real time some important operational parameters giving a detailed view of what happens to a truck while it performs various operations, and then use this information for further analysis. Contemporary data mining methods developed by researchers in this field can be applied to the collected data, to improve the operational processes, with the particular purpose of minimizing truck wear and damage risks while maximizing the production.

In preceding research, artificial neural network models were used in the experiments pursued as attempts to predict the truck rear suspension pressures in hauling loaded conditions [23]. Rear suspension pressures are crucial parameters in the truck wear process, and understanding the dynamics of fatigue accumulation can prevent expensive vehicle crashes and repairs. Proper maintenance and operation of equipment is essential to effective mining operation, so it is important to know how the pressure in these critical points depends on the changes in payload weight, truck speed, and other operational parameters such as machine rack and pitch. These parameters constantly change due to conditions of the road, its usage, weather, and the human factor involved in operation of the trucks, e.g. driving styles and habits of different operators.

Another important aspect of the mining process is a truck driver's exposure to vibration and jolt during the loading and hauling operations over rough terrain. Prediction of accelerations generated during this process is of a paramount importance. This research is targeted to gaining knowledge allowing the company to improve efficiency and personnel operational safety, while reducing

stress-related risks of its mining operations. The developed predictive models will facilitate further understanding and control of the overall driver's exposure to vibration.

The model should be capable to deal with a wide range of input data corresponding to different road surface conditions and different weather conditions. Another important element of the input data is ground speed of the truck, which is a controllable parameter. The model must be able to analyze the changes of the speed and determine the values corresponding to the safest operation at given payload, road surface, and weather conditions. A quantitative model able to directly and clearly suggest changes in ground speed of a truck in a real time operation would help to significantly reduce frequency of excessive stresses and destructive vibrations arising in major truck components. This, in turn, would reduce the downtime of equipment and lower the operating costs while improving safety of the people involved in the hauling operations.

1.3. Objectives of the thesis

In recent decades, the scientific and research community accumulated rich experience in building the theoretical foundations and experimenting with numerous practical methods of data analysis. This experience allowed many companies and organizations to extract real benefits from their data repositories and improve their processes and operations. The objectives of this research are the following:

- explore different approaches to extracting knowledge out of the available data;
- build the data analysis models on the theoretical basis of these approaches;
- compare performance of different groups of models on the example of a particular dataset in terms of prediction error (model output versus target outputs).
- define the type of model providing the best performance on this particular dataset.

The working dataset was created from a collection of raw data recorded during a truck loading and hauling operations at an oil sands mining site. To accomplish the objectives, the experiments were conducted with a number of different data analysis and prediction methods, applying them to the working dataset. Comparative analysis of different models built on the basis of these methods evaluates their performance and defines the best performing model.

In the dataset obtained at the mining site, the floor and seat accelerations are prediction targets. The three components of the floor and seat acceleration, X, Y, and Z, corresponding to each of the three spatial dimensions, were recorded using tri-axial accelerometers plugged into the truck data acquisition computer system. Their values are recorded at 13 Hz by the monitoring equipment, to be analyzed as the dependent variables. The specific target of this research work is to define a model, or group of models, that are capable to approximate most adequately, and with as much precision as possible, the floor and seat acceleration as a function of the operating parameters present in the available working dataset. The successful model must be able to analyze the historical dynamics of these accelerations and predict their behavior based on anticipated combinations of the input parameters at different moments of time.

1.4. Approach and methodology

All the methods applied to the working dataset can be grouped into two categories. The first category embraces artificial neural network models analyzing the entire dataset. Neural networks are well described in literature [2], and proven to be universal approximators able to learn and model any continuous function with any desired accuracy. They are capable not only to represent complex non-linear functions, but also to adapt to changes in modeled systems. It is worthwhile to experiment with neural network models since they have a proven history of success and often deliver good results.

The second category uses a collection of local models each built and trained on its proper cluster of the data. The rationale is to cluster the whole dataset into the groups of data that are similar to each other within each group. In this case, each of the local models adjusts its parameters according to the specific properties of the data belonging to their designated cluster, thus describing the data in each cluster more precisely than general model built for the entire dataset. Different methods of implementation of the local models approach were tried and compared to each other, such as linear models, Takagi – Sugeno model [8], and RBF networks.

In each case the working dataset is divided into training and testing parts in proportion 2:1, each model is built and tuned on the training dataset, then evaluated on the testing data. To evaluate and compare the performance of the models on the testing dataset, the outputs produced by them are compared to the real outputs existing in the dataset. In this way, the testing part of the dataset imitates the future data not yet encountered by a model, at the same time providing a means to evaluate the model performance on this data against targets (existing outputs).

1.5. Literature review

The source and nature of the data analyzed in this research are described in [23], so are the research results obtained for the truck rear suspension pressures as a function of the operational parameters monitored during several cycles of operation under loaded hauling conditions.

The theoretical and practical aspects of data mining methods and approaches are discussed in [3], which includes abundance of detailed information and recommendations on implementation of the described methods into practice. In particular, the book proposes the algorithms and formulas for data preprocessing, FCM clustering method, and RBF network implementation.

Neural networks theory is presented in [1], [2], [3], [4], [5], [18], [19], [20], [21], [22], with detailed description of back-propagation algorithm and gradient descent in [4], [5], [18], [19], [21]. The most popular neural network architecture, feed-forward network with one hidden layer, was first proposed in [21], and discussed at length in most neural network textbooks, including the sources listed above.

[5] proposes the stages of neural network design, with some practical recommendations on the number of neurons. The use of neuron transfer functions other than sigmoid, as well as some other practical aspects of efficiency of the back-propagation algorithm, is discussed in [19]. Computing partial derivatives in the gradient descent method, and the influence of the learning rate on the neural network behavior are explained in detail in [4].

For the data representing time series, [22] asserts superior performance of the neural network models when compared to the conventional linear time series and regression models, and emphasizes the importance of selecting properly the subset of input variables for the model prediction ability.

The idea of data partition into clusters based on similarities is discussed in [3], [6], [7], [8], [9], [10]. The clustering methods such as k-means and FCM (Fuzzy C-Means) were proposed in [11] and [12], and practical implementation of the FCM algorithm was described in detail in [3], [7], and [9]. Further tuning of the method through computing the covariance matrices between data points and cluster prototypes was proposed in [10] and described as GK (Gustafson-Kernel) algorithm.

Theoretical underpinnings and implementation of the linear regression model, with derivation of the least square solution formula, are described in [13], [14], [15].

The Takagi – Sugeno model was introduced by authors in [8] as a mathematical tool to build a fuzzy model of a system using fuzzy implications and reasoning. To partition the space of premise variables into a set of fuzzy subspaces and to determine the membership functions of the fuzzy sets in the premises, Takagi and Sugeno proposed a premise parameters identification algorithm. Their algorithm consisted of a series of consecutive heuristic search procedures, at each step dividing the input space into "big" and "small" areas along each dimension. In a later work [9], Sugeno and Yasukawa improved the premise parameters identification phase of Takagi – Sugeno model by introducing, for this purpose, the FCM (Fuzzy C-Means) clustering algorithm [12].

The realization of radial basis functions within the framework of neural networks was proposed in [17] and [11]. Radial basis function (RBF) networks are described in [3], [5], [6], [7], [16], [17], with the detailed mathematical description of their operation in [6]. Some of the possible basis functions, besides the most popular of them, Gaussian function, are presented and discussed in [3] and [6].

The ideas, approaches, and recommendations retrieved from the analyzed literature allowed us to create a framework for analysis of the particular working dataset, and to build a number of specific data analysis models within this framework. Some of the methods described in the literature, such as linear regressions, feed-forward neural networks, FCM clustering, and RBF networks, are implemented directly in the software-based models built particularly for this research. Other ideas and their combinations are used to modify the described methods and to build different models, such as neural networks with feedback from previous data points, Takagi-Sugeno models based on FCM clustering, and RBF networks based on functions other than Gaussian. The subject of interest is comparative analysis of all of the implemented models, to evaluate their performance with respect to each other and to define which of them was able to describe the working dataset best of all and predict the outputs in the most precise way.

Chapter 2. Description of the data analyzed

This chapter describes the nature of the data analyzed in this research, the choice of parameters considered inputs and outputs, the process of data collection at the industrial operations site, and the sequence of steps to obtain the working dataset from the raw data. Also, the chapter presents some statistical information about the dataset, as well as plots of the inputs and outputs. Possible models of data analysis and prediction are discussed and classified. Some attempts of data visualization are made with the aim to facilitate the definition of the model parameters.

2.1. Industry background

Over the past 20 years, the oil sands industry in Canada has grown substantially with the recent conceptual modifications of the operational processes (shift from draglines and bucketwheel reclaimers to the shovel-truck methods in primary mining and handling of materials). Large haul trucks with capacities over 400 tons (Figure 2.1) became the main hauling means in the mining operations. This led to an increased awareness of truck reliability, since, as the equipment becomes larger and transports heavier loads, the amount of stress on its components increases too.

Proper exploitation and maintenance of equipment is essential to effective mining operation. To cut down maintenance costs and to ensure the highest possible utilization of equipment, there is a need to evaluate the varying effects of driving conditions, loading and dumping on the level of stress in the truck components. Proper conditions of such critical components as frame, suspension cylinders, rims, and tires can extend the total life of a machine. That is why the knowledge about their conditions is of such a great importance.

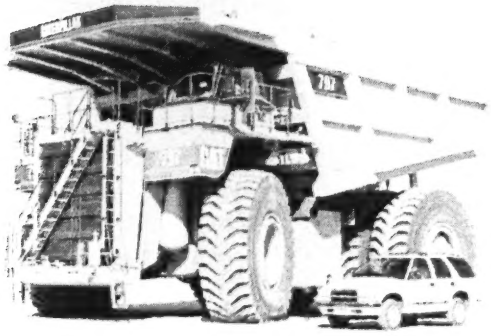


Figure 2.1 The Caterpillar 797 haul truck

A mining company, one of the largest oil sands producers, operates an open pit mine in northern Alberta, Canada. The latest generation of its haul trucks is outfitted with sophisticated monitoring systems [23]. The operational goal here is to minimize truck damage while maximizing production.

The move to truck and shovel mining operations has led to an increased awareness of truck reliability [23]. The evolution of the truck and shovel operations has had a major effect on cost reduction but, as the equipment becomes larger, the equipment components are subjected to much higher dynamic stresses. A record of real time parameters gives a detailed view of what happens to a truck as it performs various functions. The input parameters of great importance are payload, ground speed, rack and pitch values of the machine. By knowing how these values influence the floor and seat acceleration, it is possible to assess the remaining fatigue life of the truck frame.

Damage frequency is exponentially dependent on payload and provides a measure of how fast the components are depreciated, based on an understanding of fatigue [23]. Component life in a payload spectrum enables cost calculation. The minimum unit production cost yields the optimal payload for a truck to haul. To effectively implement the analysis, continuous monitoring of truck and shovel parameters is needed, such as suspension cylinders pressure, ground speed, payload weight, rack and pitch [23]. Continuous monitoring means real time values are registered every second or even more frequently.

2.2. Methodology and raw data collection

The data were obtained from field tests of the Cat 797 (Caterpillar) truck traveling loaded from electric shovel pit to the dump. The input data for the neural network algorithm are collected using the Vital Information Management System (VIMS) developed by Caterpillar [23]. VIMS includes on-board truck measuring equipment and off-board VIMS software, which enables data download to a laptop computer. The VIMS Data Logger may be activated to record real time data that can be downloaded, uncompressed and exported to Excel.

By monitoring the Cat 797 haul trucks and collecting the data from the VIMS, one is able to indirectly analyze the stresses on the truck frame as a function of different input data, such as ground speed, time, haul route location, payload weight, rack, pitch, and effective grade.

The frame of the Cat 797 truck is connected to the suspension system. The suspension consists of four hydraulic cylinders attached to the front and rear of the truck frame. The front cylinders are bolted to outward facing panels, while the rear cylinders are attached to lower holes in the rear arms of the frame (Figure 2.2) [23].

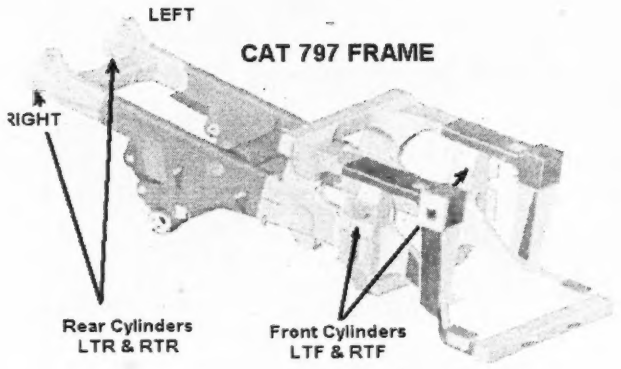


Figure 2.2 Suspension cylinder to frame attachment locations

Pressure in the four hydraulic cylinders is used to obtain a measure of frame deformation known as rack [23]. It seems reasonable to assume that deterioration of the frame is similarly connected to deterioration of the cylinders, linkages and axle box because these are all connected together. At any rate, the cylinders are the basis for gaining a measure of how the various components move around and thus deteriorate.

Rack occurrences are most responsible for frame damage [23]. Rack is twisting of the frame expressed in pressure units, i.e. it can be seen as a pressure approximation of frame deformation. A rack value is recorded once every second by VIMS. Rack is calculated directly from pressures in the four suspension cylinders attached to the frame. The sign of the rack value indicates the direction of the twist.

Another measure of frame deformation recorded by VIMS is known as pitch, expressed in pressure units as well. Pitch is a measure of front to back rocking motion, like driving over speed bump [23]. Pitch also contributes damage to the frame, but not as severely as rack.

2.3. Importance of data preprocessing

Preprocessing is a sequence of operations converting raw data (measurements, signals, images, etc.) or other objects' characteristics, into a data representation suitable for processing tasks (like prediction) [3]. The preprocessing sequence depends on the data type and on the processing goals, however, it generally includes the following steps:

1. Collection and storage of data and knowledge about a domain
2. Analysis of available data from the point of view of processing goal
3. Basic preprocessing of raw data
4. Feature extraction / creation from raw data
5. Pattern forming for defined objects and pattern encoding
6. Forming an information system (dataset in a tabular form)
7. Basic preprocessing of the information system
8. Feature extraction / creation from the information system
9. Feature selection and evaluation, reduction of dataset

Preprocessing of raw data is done before feature extraction and pattern forming in order to clean, enhance and initially transform data into a form best suitable for further processing [3]. The character of preprocessing of raw data heavily depends on data type. For a dataset already containing patterns in a raw format, preprocessing operations may include validation, detection and handling of outliers and missing data, averaging, scaling and normalization.

Feature extraction provides a better (possibly shorter) object representation most suitable for a given processing goal [3]. Features can be selected from raw or preprocessed data. Then patterns are formed from raw or preprocessed data or from extracted features. Pattern forming requires defining an object for a given dataset and processing type, and then a pattern can be formed as a

characteristic description of an object. For preprocessed data, an information system can be constructed in a tabular form of object patterns paired with associated desired target output values (for supervised learning/design), which is called a decision table [3]. The information system, in turn, can be additionally processed through validation, completeness and consistency testing, outliers discovery and handling, encoding nominal attributes, discretization of continuous attributes, scaling and normalization of the attributes.

Scaling is performed for each attribute separately according to the formula [3]:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{2.1}$$

where the scaled attributes will take on values from the interval [0,1].

Normalization of an attribute guarantees that the normalized attribute will have zero mean and unit standard deviation. It is also performed for each attribute separately according to the formula [3]:

$$x_{normalized} = \frac{x - \mu}{\sigma_x} \tag{2.2}$$

where μ is the mean for the attribute x and σ_x is the standard deviation of the attribute x.

2.4. Creating a working dataset from the original raw data

The raw data registered on the mining site were obtained for analysis in the form of six Excel workbooks. Each workbook represented one working cycle of loading the truck, about 20 minutes long, and consisted of two spreadsheets containing the following measurements (fragments are shown in Table 2.1 and Table 2.2):

Table 2.1 Fragment of the first spreadsheet of the raw data workbook 103num01cr.

VIMSTime_Date	18/07/2001	18/07/2001	18/07/2001
VIMSTime_Time	11:05:01	11:05:02	11:05:03
Time	11:06:11	11:06:12	11:06:13
Ground Spd (719)	9.66	11.27	12.07
Haul Distance (730)	2.25	2.25	2.25
Machine Pitch (792)	188	472	-283
Machine Rack (793)	-566	-282	-659
Payload (728)	0	0	0
Susp Cyl LTF (713)	3016	3111	2922
Susp Cyl LTR (714)	3205	3016	3393
Susp Cyl RTF (716)	3205	3205	3016
Susp Cyl RTR (715)	2828	2828	2828
Actual Gear (348)	1.00	2.00	2.00
Machine Roll	188	94	471
Max Roll Change	377	377	0
Max Rack Change	377	377	0
Max Pitch Change	755	755	0

Table 2.2 Fragment of the second spreadsheet of the raw data workbook 103num01cr.

Time	11:04:14	11:04:14	11:04:14	11:04:14	11:04:14
Flr X accel	0.0143	0.0102	-0.0058	0.0093	-0.0096
Flr Z accel	1.0805	1.0848	1.0976	1.0726	1.073
Flr Y accel	-0.2106	-0.2143	-0.2594	-0.267	-0.2792
Seat X accel	-0.1685	-0.1658	-0.1396	-0.164	-0.1337
Seat Y accel	-0.1122	-0.1138	-0.0793	-0.0549	-0.0617
Seat Z accel	1.1489	1.1544	1.1582	1.1597	1.1367

The first spreadsheet of the workbook 103num01cr contained 1,200 VIMS data measurements taken at 1 Hz, i.e. every second. The second spreadsheet contained about 15,000 acceleration data measurements taken at 13 Hz, i.e. 13 times per second.

The other five workbooks were organized in the same way and each contained more or less the same amount of raw data.

Eight major measurements from the first spreadsheet of each workbook were chosen as the most informative to be interpreted as input attributes (Table 2.3):

Table 2.3 Data measurements interpreted as inputs

Input #	Parameter measured	Measurement units
1	Ground speed of the truck	Miles per hour
2	Machine pitch	Pounds per square inch
3	Machine rack	Pounds per square inch
4	Payload weight	Thousand pounds
5	Pressure in suspension cylinder LTF	Pounds per square inch
6	Pressure in suspension cylinder LTR	Pounds per square inch
7	Pressure in suspension cylinder RTF	Pounds per square inch
8	Pressure in suspension cylinder RTR	Pounds per square inch

All six measurements from the second spreadsheet were interpreted as outputs, i.e. labels assigned to each data point (Table 2.4). They are measured in multiples of g (gravity acceleration):

Table 2.4 Data measurements interpreted as outputs

Output #	Parameter measured	Measurement units
1	Floor X acceleration	Multiple of g
2	Floor Y acceleration	Multiple of g
3	Floor Z acceleration	Multiple of g
4	Seat X acceleration	Multiple of g
5	Seat Y acceleration	Multiple of g
6	Seat Z acceleration	Multiple of g

To create a single dataset available for analysis, the six original Excel workbooks were concatenated into one as a sequence. Then the outputs needed to be matched to the inputs according to their time-stamps. As the acceleration data were measured with a higher frequency (by factor of 13) compared to the input data measurement, the original data required preprocessing. The preprocessing procedure completed the following sequence of steps:

1. Transform time stamps for inputs and outputs from Excel time format into floating point numeric format (for example, a time stamp 11:04:14 becomes represented as 0.46127).
2. Round all the numeric time stamps, both for inputs and outputs, to 5 digits after the floating point, so that for each input data point of format (time, x1, x2, ..., x8) there were about 13 corresponding output data points of format (time, y1, y2, ..., y6) with identical time stamp.
3. For each input data point in the first spreadsheet (Table 2.1), find in the second spreadsheet (Table 2.2) all corresponding output data points with identical numeric time stamp and compute their mean values for each of the outputs y1, y2, ..., y6.
4. Assign the obtained mean values to the outputs corresponding to this particular data point.

Thus, the preprocessing procedure computed the average value of each output for the 13 data points from the second spreadsheet. Then it assigned this average value to the corresponding (by the time stamp) output of this particular data point from the first spreadsheet. This way, all 13 output data points matching each specific input data point were averaged and assigned as one resulting value to this input to produce a single data point.

In each working cycle, measurements began to be registered before the first load, i.e. on the empty truck (Figure 2.3). Thus, the last preprocessing step needed to remove all data points with zero

payload from the resulting dataset, since they did not represent any interesting information to be analyzed.

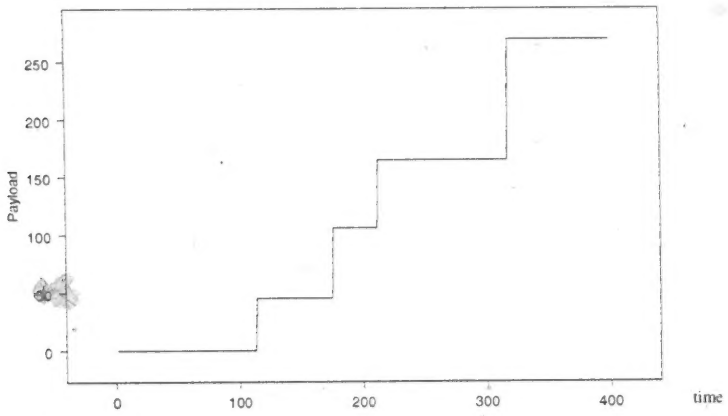


Figure 2.3 Fragment of the raw dataset representing increase of the truck payload measured during the beginning of one of the working cycles

As a result, pre-processed working dataset contained 3,170 data points concatenated from six time series representing six loading cycles of one truck, each about 20 minutes long. Figure 2.4 and Figure 2.5 plot the inputs of the resulting dataset, while Figure 2.6 and Figure 2.7 plot the outputs, where the horizontal axis represent seconds (data points were registered every second). The first 2,155 data points (68%), representing the first four working cycles, were used for training. The last 1,015 data points (32%), representing the two remaining working cycles, were used for testing.

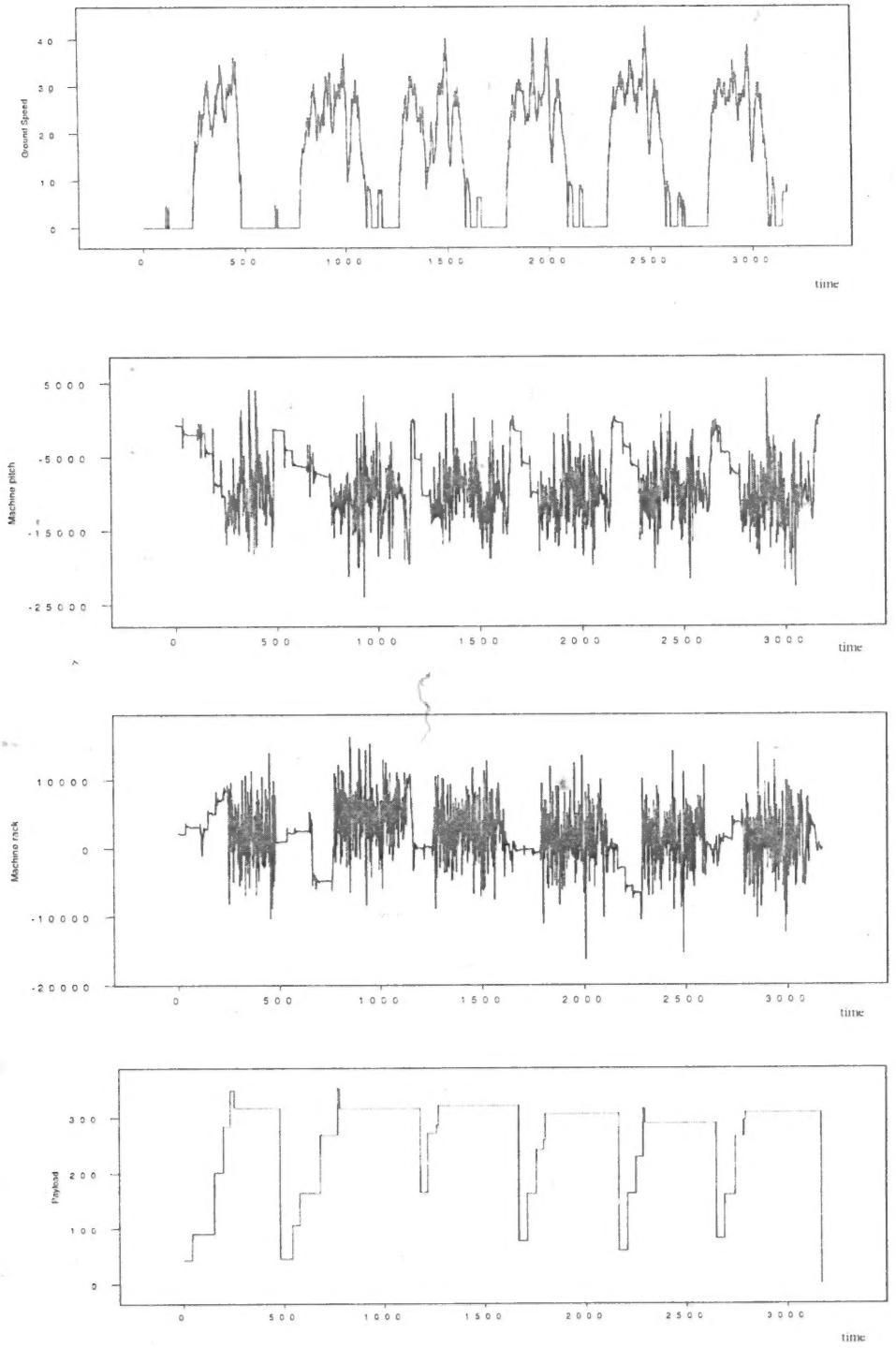


Figure 2.4 Inputs 1 to 4 of the working dataset: ground speed, machine pitch, machine rack, and payload

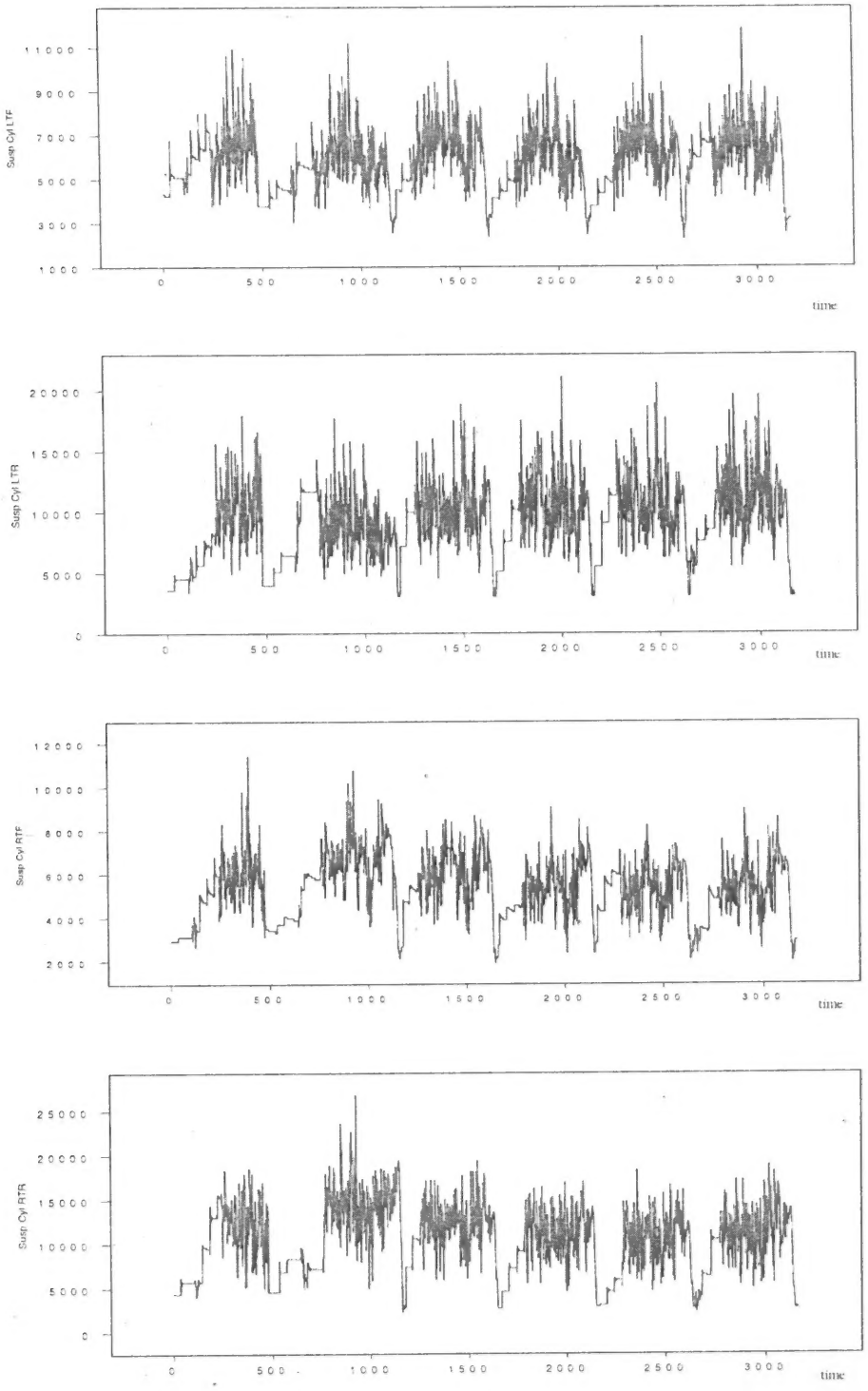


Figure 2.5 Inputs 5 to 8 of the working dataset: pressure in each of the 4 suspension cylinders

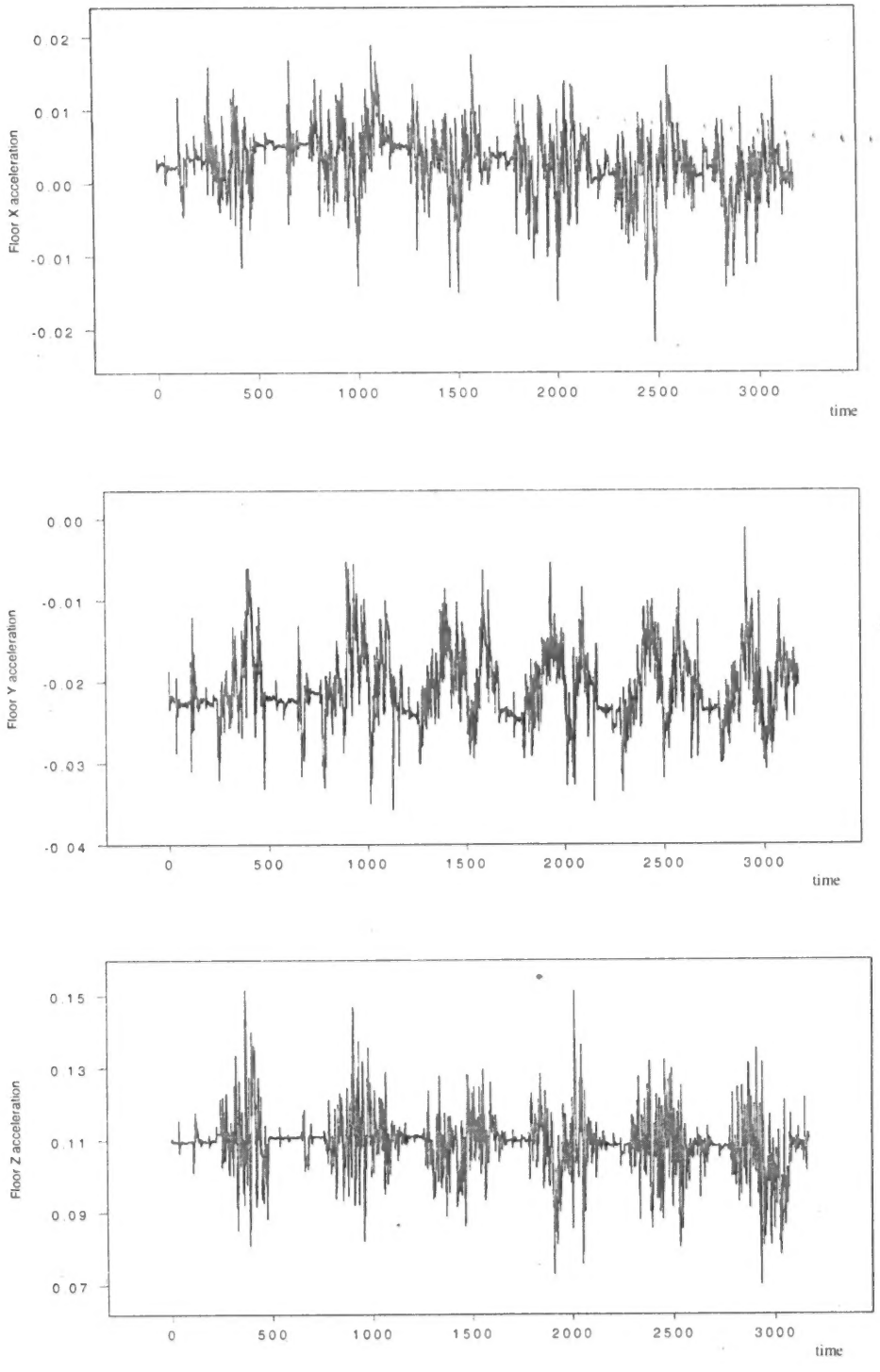


Figure 2.6 Outputs 1 to 3 of the working dataset: X, Y, Z components of the floor acceleration

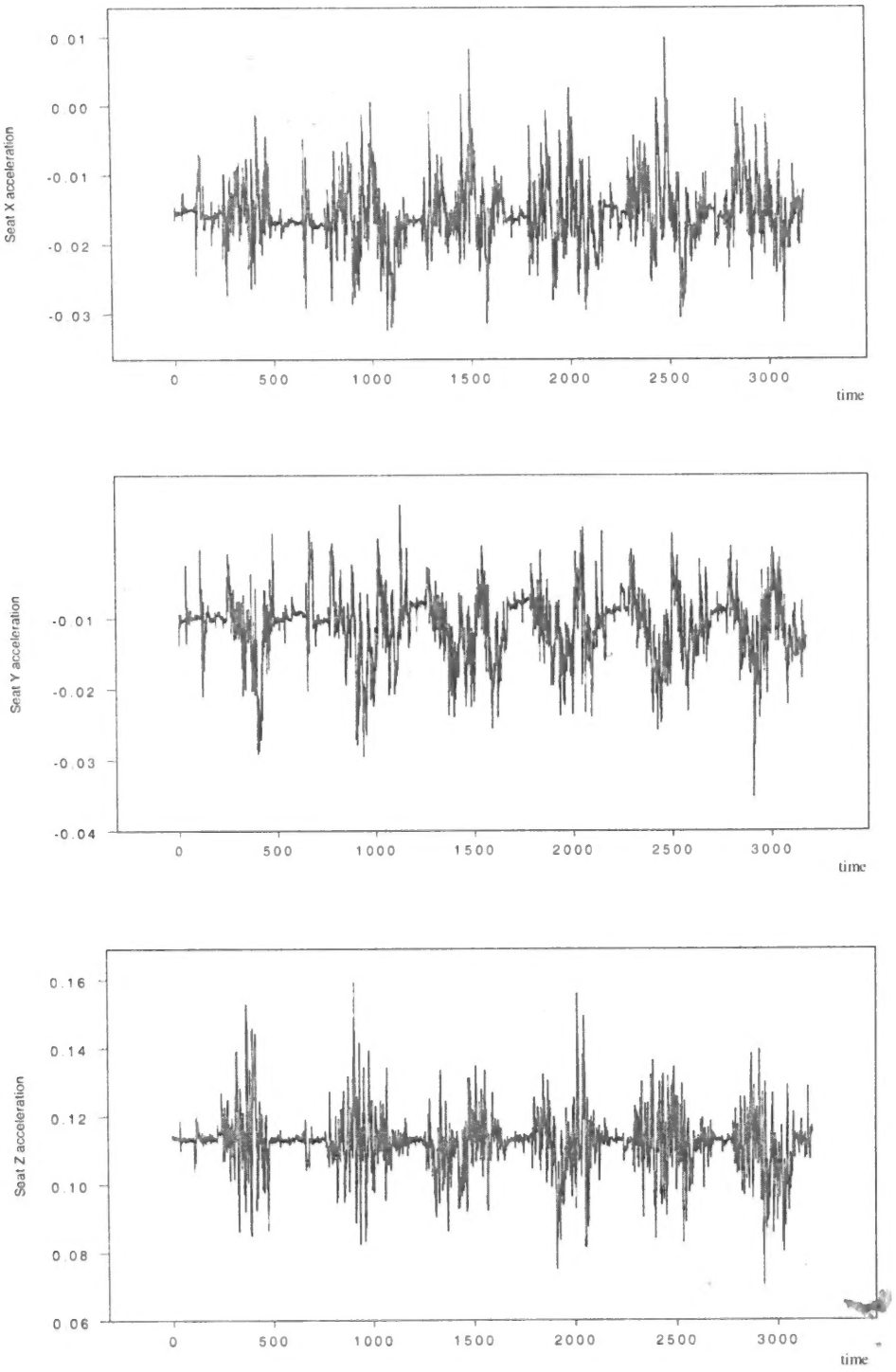


Figure 2.7 Outputs 4 to 6 of the working dataset: X, Y, Z components of the seat acceleration

2.5. Statistical information about the working dataset

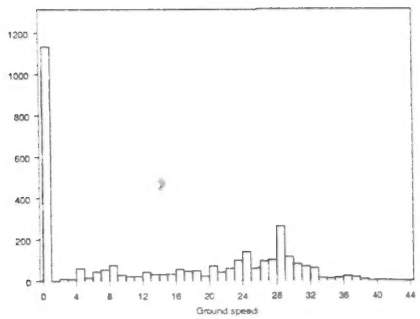
The analysis of the general statistical properties of the working dataset (Table 2.5, Table 2.6, Figure 2.8, and Figure 2.9) may give some insight into understanding of the data. This knowledge can probably serve as a starting point in determining the parameters of the model to be built for the data analysis.

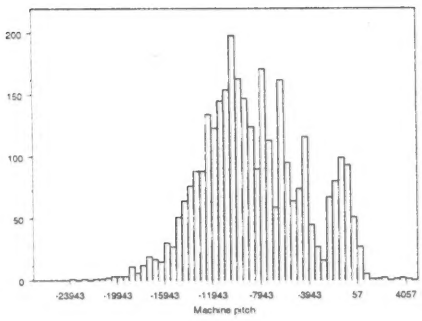
Table 2.5 Minimum, mean, maximum values and standard deviation for each of the data inputs and outputs throughout the entire dataset (3170 data points)

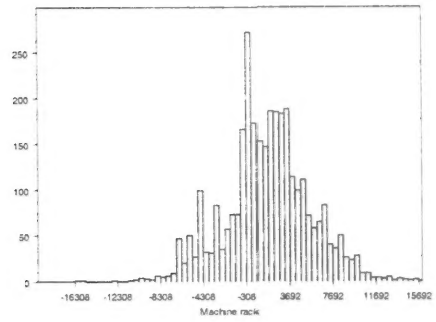
Data inputs and outputs	Min value	Mean value	Max value	Std deviation
Input 1: Ground speed of the truck	0	14	43	13
Input 2: Machine pitch	-23943	-8395	5562	4328
Input 3: Machine rack	-16308	1643	16213	3956
Input 4: Payload weight	0	263	354	84
Input 5: Pressure in susp. cylinder LTF	2262	5793	11877	1354
Input 6: Pressure in susp. cylinder LTR	3016	9169	21115	3011
Input 7: Pressure in susp. cylinder RTF	1885	5307	11406	1439
Input 8: Pressure in susp. cylinder RTR	2262	10326	26771	3848
Output 1: Floor X acceleration	-0.021775	0.002743	0.018975	0.004428
Output 2: Floor Y acceleration	-0.035760	-0.020911	-0.001225	0.004328
Output 3: Floor Z acceleration	0.069885	0.108978	0.151465	0.007906
Output 4: Seat X acceleration	-0.032355	-0.015640	0.009855	0.004649
Output 5: Seat Y acceleration	-0.035220	-0.011038	0.006100	0.004641
Output 6: Seat Z acceleration	0.070245	0.112154	0.159185	0.008242

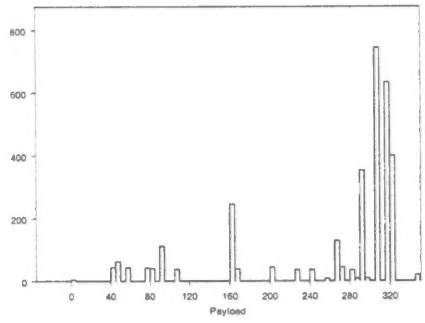
Table 2.6 Correlation matrix between inputs and outputs of the working dataset

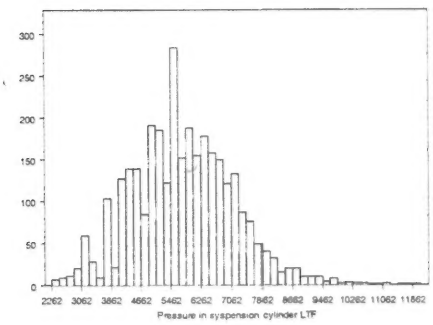
Correlation	Output 1	Output 2	Output 3	Output 4	Output 5	Output 6
Input 1	-0.25458	0.294526	-0.06191	0.220491	-0.26929	-0.06324
Input 2	0.030031	0.072746	0.051033	-0.05154	-0.09472	0.049424
Input 3	0.172282	0.090166	0.000833	-0.13586	-0.05866	-0.0257
Input 4	-0.03232	0.248885	-0.03018	0.021014	-0.20442	-0.05077
Input 5	-0.29696	0.34354	-0.04134	0.246035	-0.35041	-0.04385
Input 6	-0.26831	0.042972	-0.05582	0.236941	-0.05096	-0.03836
Input 7	0.317541	0.341268	-0.01206	-0.32803	-0.2521	-0.05177
Input 8	0.190394	0.133031	-0.03278	-0.16349	-0.07114	-0.06037

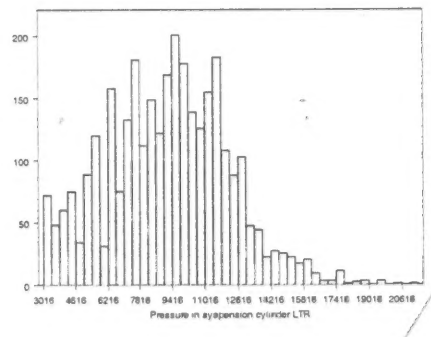

1200
1000
800
600
400
200
0
0 4 8 12 16 20 24 28 32 36 40 44
Ground speed

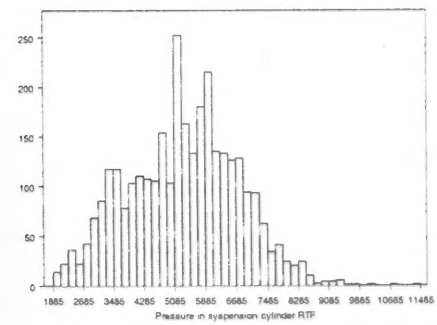

200
150
100
50
0
-23943 -19943 -15943 -11943 -7943 -3943 57 4057
Machine pitch


250
200
150
100
50
0
-16308 -12308 -8308 -4308 -308 3692 7692 11692 15692
Machine rack


800
600
400
200
0
0 40 80 120 160 200 240 280 320
Payload


300
250
200
150
100
50
0
2262 3062 3862 4662 5462 6262 7062 7862 8662 9462 10262 11062 11862
Pressure in syspension cylinder LTF


200
150
100
50
0
3016 4616 6216 7816 9416 11016 12616 14216 15816 17416 19016 20616
Pressure in syspension cylinder LTR


250
200
150
100
50
0
1885 2685 3485 4285 5085 5885 6685 7485 8285 9085 9885 10685 11485
Pressure in syspension cylinder RTF

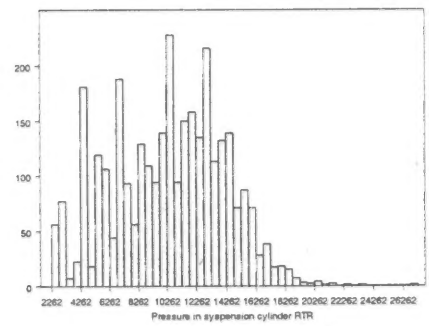

200
150
100
50
0
2262 4262 6262 8262 10262 12262 14262 16262 18262 20262 22262 24262 26262
Pressure in syspension cylinder RTR

Figure 2.8 Histograms of the inputs

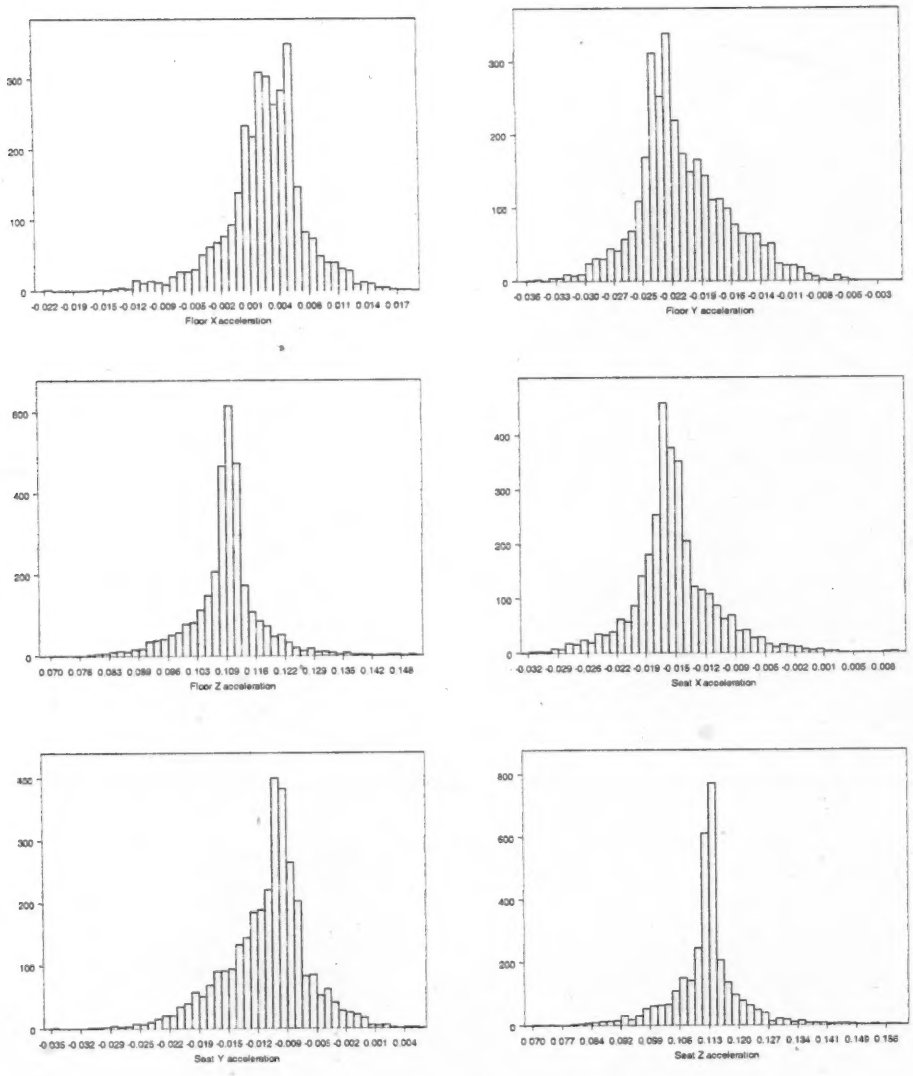


Figure 2.9 Histograms of the outputs

All the outputs (floor and seat accelerations) have clearly expressed normal distribution around approximately 0.1g for the vertical component (Z), and about an order of magnitude lower values for the horizontal components (X and Y). As for the inputs, we can say that machine pitch and rack, as well as the cylinder pressures, have a tendency to normal distribution, while ground speed has a peak at 0, and the most frequent payload value is around 300 thousand pounds.

From the correlation matrix for the entire dataset (Table 2.6), the best correlation coefficient 0.35 is between the input x5 (Susp Cyl LTF) and the output y5 (Seat Y accel). Also, there is a somewhat similar level of correlation (0.22 to 0.35) between the inputs x1 (Ground Speed), x5 (Susp Cyl LTF), x7 (Susp Cyl RTF) from one side, and the outputs y1 (Floor X accel), y2 (Floor Y accel), y4 (Seat X accel), y5 (Seat Y accel) from the other side. Other than that, the correlation between inputs and outputs is quite weak. We can see that outputs y3 (Floor Z accel) and y6 (Seat Z accel) do not correlate at all with any of the inputs: coefficient values are from 0 to 0.06. Also, input x2 (Machine Pitch) does not correlate with any of the outputs (none of the coefficients exceeds 0.1).

2.6. Possible models of data analysis and prediction

The truck and shovel operations can be represented by the changes of parameters, such as ground speed, rack and pitch of the haul truck, payload weight, and suspension cylinders pressure during an operational cycle. To understand how these operations affect the wear and fatigue of the truck frame and influence the frequency of potential damages, the temporal acceleration parameters are measured at the driver's seat and on the floor of the driver's cabin. These accelerations can be directly translated into the physical stresses applied to the frame during the operational cycle. In that sense, it is interesting to explore the relationships between the listed above operational parameters as arguments from one side, and each of the acceleration parameters as a function of these arguments from the other side.

To be able to explore these relationships, a model needs to be built, that would accept the truck and shovel parameters as inputs and transform them into the acceleration parameters as outputs (Figure 2.10). The model should be able to learn from historically collected data and predict the outputs as close to the targets as possible. This research is aimed at looking for a number of possible models that can be built to approximate the functions describing these relationships.

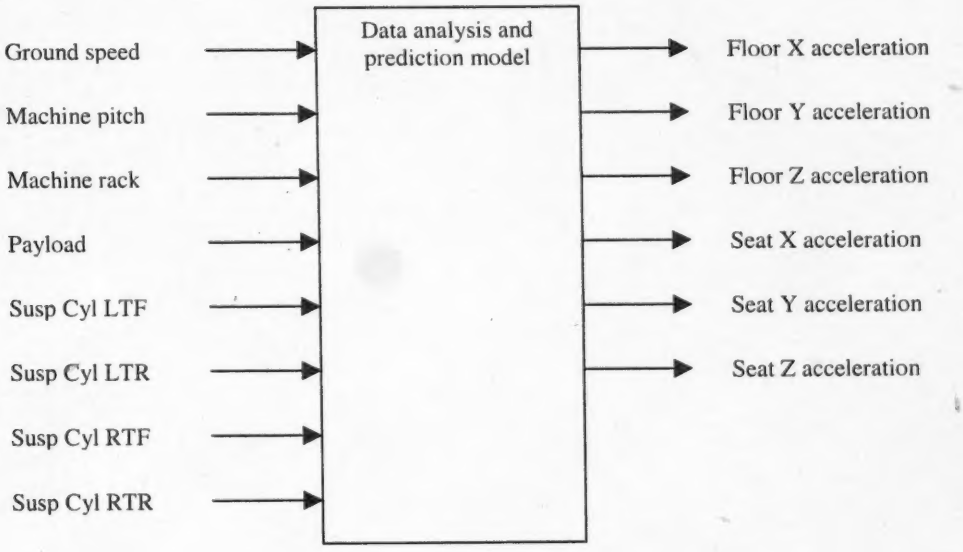


Figure 2.10 Model built to predict the acceleration parameters from truck and shovel parameters

Two types of models are considered in this research: models based on artificial neural networks analyzing the working dataset in its entirety, and local models based on clustering data into the groups exhibiting some similarities within each of them. Artificial neural networks are popular due to their ability to learn from existing data and generalize the acquired knowledge to unseen data. Multi-layer feed-forward neural networks are known as universal approximators, theoretically able to approximate any continuous function to any extent of accuracy [2].

As for local models based on the data clusters, the approach is to divide the data into groups based on predefined criteria of similarity. Rather than to apply a universal model to the entire dataset, a separate model with its proper parameters is used for each data cluster. The parameters of each of the models are locally tuned to make the models work as precisely as possible for their respective clusters of data.

In general, the less size are the clusters of the data from which a model learns, and the more homogeneous are the data within each cluster, the more precisely the local models are able to describe them. However, with further increasing the number of clusters, each local model would describe only small portion of the data and there would be too many such models. The aim here is to have as few local models as possible that are capable of providing better overall performance on the testing data.

All of the described models, where acceleration parameters are considered functions of inputs at every particular moment of time, can be perceived as static models. Besides that, the experiments were conducted on the models having dynamic nature. In these models, in addition to the inputs listed above, the arguments of another sort were introduced into the model. These additional arguments represented the target outputs corresponding to a preceding data point, or even several preceding data points. In that sense, each of the outputs of the model could be considered not only a function of inputs, but also a function of time component, since the outputs are defined not only by inputs at each particular moment of time, but are also affected by the target outputs that existed a moment or several moments of time ago.

2.7. Conclusions

The working dataset, compiled from raw data and preprocessed to be fed into the model, represents six consecutive operational cycles of loading a haul truck and its travel under the load. Floor and seat accelerations were chosen as dependent variables, and their projections on each of three spatial dimensions constitute the targets to be approximated by a model. All the accelerations are normally distributed, and most of the independent variables have normal distribution too, except ground speed and payload inflated with zeros. For that reason, data points with zero payload were excluded from consideration as not interesting. Preprocessed dataset thus is ready to be presented to the models of different types discussed in detail in following chapters.

Chapter 3. Artificial neural networks

This chapter describes the theoretical foundations of neural networks and presents the results of practical experiments with the models built on the basis of neural networks approach. For each experiment and each model, the performance index is calculated for training and testing parts of the dataset, to be able to compare the results delivered by different models. In particular, the following parameters of the neural network were modified, to explore their influence on the model performance:

- number of neurons in the hidden layer
- number of learning epochs
- learning rate
- momentum
- type of neuron transfer function in the hidden layer
- linear vs. non-linear output layer transfer function
- two hidden layers vs. one
- feedback from previous cycle target outputs

3.1. Neural Networks Foundations

3.1.1. Neurobiological analogy

The human nervous system, from engineering perspective, consists of three conceptual layers: receptors, processing unit, and effectors. The processing unit – the brain – receives electrical impulses from receptors, processes them, and generates electrical impulses for the effectors. Technically, the human brain can be considered a massively parallel distributed processor made up of elementary processing units. These units called neurons convert electrical signals into chemical signals and back. The brain has an enormously complex structure: it consists of approximately 10 billion neurons, and 60 trillion connections between them [1].

It is important to recognize that such a complex structural organization is a unique characteristic of the brain and that the most advanced artificial neural networks we are able to design are truly primitive and cannot be even remotely compared with the circuits found in the brain [2]. However, remarkable progress was made in this field over the past two decades. Neural network research is motivated by two desires: to obtain a better understanding of the human brain and to develop computers that can deal with abstract and poorly defined problems [4]. For example, understanding speech and recognizing faces represent trouble for computers, but humans do extremely well at these tasks. The neural network concept exploits analogy with the human brain. An artificial neural network is an adaptive machine designed to model the way in which the brain performs a particular task or function of interest [2]. It imitates brain in two aspects:

1. Knowledge is acquired from environment through a learning process.
2. Inter-neuron connections, or synaptic weights, store the acquired knowledge.

The learning process exploits an algorithm that modifies the free parameters of the network (synaptic weights) in an orderly fashion to attain a desired design objective [2]. The supervised learning paradigm involves modification of the synaptic weights by analyzing a set of labeled

training samples (that can be considered a teacher), each consisting of a unique input signal and corresponding target output. After the network is presented with an example from the set, its synaptic weights are modified to minimize the difference between the target response and the network's actual response. The training is repeated until the network reaches a stable state when there are no more significant changes in its synaptic weights. At this point we can say the system has learnt the knowledge from that set of labels representing a teacher, in a way conceptually similar to what a human brain does when learning from the external world.

3.1.2. Principles of learning process

The ability of a neural network to provide useful data manipulation comes from its ability to properly determine the connection weights between neurons during the learning process. This is a dramatic departure from conventional information processing where solutions are described in step-by-step procedures [4]. A conventional information processing system would approach a classification problem with appropriate mathematics and complicated algorithms, such as correlation and frequency spectrum analysis. With a neural network, a set of several hundred or thousand data examples is fed into the input layer, resulting in values generated by the output layer. By selecting the proper weights, the output can be configured to report a wide range of information.

The term learning is widely used in the neural network field to describe this process. A learning algorithm can be defined as a process of determining an optimized set of weights based on the statistics of the examples [4]. Regardless of the specific features of the method, the resulting weights are virtually impossible for humans to understand. Patterns may be observable in some rare cases, but generally they appear to be random numbers. In this sense, the neural network approach can be considered "black box", since there is no reasonable answer for why this particular set of weights works for this particular dataset. The training dataset is fed into the

system, then the system tunes itself in an unpredictable way, and finally offers a solution. The solution is often quite acceptable, but yet impossible to interpret or explain.

The learning algorithm improves the performance of the network by gradually changing each weight in the proper direction, through an iterative procedure. At each iteration (called an epoch), the learning algorithm goes through all the weights and makes the whole system slightly better in approximating the target data. The iteration loop is repeated either for a specified number of epochs, or until it reaches some termination criteria where no further improvement is being made from iteration to iteration. That means the phase of learning from an existing data set is completed. The neural network is ready and then, with its established weights, it can be used to generate predictions for the new data.

In order for this iterative strategy to work, there must be a single parameter describing current performance of the system at each iteration. This parameter is called performance index and is usually represented by the averaged error between the target and the real output of the system throughout all data points. Most popular measures are MSE – Mean Squared Error (3.1, 3.2), and RMSE – Root Mean Squared Error (3.3):

For a single output:
$$MSE = \frac{\sum_{k=1}^{N} (target_k - y_k)^2}{N} \tag{3.1}$$

Averaged for "m" outputs:
$$MSE = \frac{\sum_{j=1}^{m}\sum_{k=1}^{N} (target_{jk} - y_{jk})^2}{N \cdot m} \tag{3.2}$$

$$RMSE = \sqrt{MSE} \tag{3.3}$$

where

- m is the number of outputs in the dataset
- N is the number of data points in the dataset

- y_{jk} is the output j of the model for the data point $\mathbf{x}_k$
- $target_{jk}$ is the target value of this particular output (already existing in the dataset and taken from there)

At the beginning of each iteration, the performance index has to be set to zero so that it can accumulate the differences between the neural network response and the real labels (target data). By the end of the iteration, the value of the performance index accumulates the errors throughout all data points. Its value be used either as a termination criterion to stop training process if there is no further change in performance, or simply for visualization of convergence process if the system runs a fixed number of iterations to be trained. Usually, the connection weights of the neural network are initialized randomly and therefore, in the first iteration, the network produces irrelevant outputs. So, the performance index starts from a high value, and then gradually converges to some lower level as the neural network learns to recognize the targets.

For verification purposes, it makes sense to reserve part of the existing dataset for testing (30 to 40%), to imitate the new data not yet presented to the system. The benefit of this approach is that the reaction of the neural network to these new data can be compared with the target data actually existing in reality. The same performance index, i.e. the difference between the neural network response and the real labels (target data) can serve as evaluation of the quality of training carried out before the system is tested.

3.1.3. Back-propagation learning algorithm

The learning rule now called the back-propagation algorithm existed under the name of the stochastic approximation method since 1951. As a technique exploiting non-linearity and differentiability of sigmoid transfer function in neural networks, it was popularized by Rumelhart and McClelland in1986 in the Parallel Distributed Processing (PDP) edited volumes [21]. The

PDP volumes summarized the neural research of several psychologists and computer scientists at the University of California and helped secure for neural networks a broad interdisciplinary audience. The back-propagation algorithm quickly dominated the neural network literature and was heralded by the technical and popular press as a long awaited breakthrough in machine intelligence [20]. However, it has been criticized as having several conceptual drawbacks:

- it converges to a local error minimum, without any guarantee that it is anywhere close to the global minimum, and it often fails to converge at all;
- little changes of the network parameters and initial conditions change the network behavior in unpredictable way, this may lead to oscillations and even chaotic wandering;
- training cycles are lengthy which takes time and often prevents the network from learning the complex datasets within a reasonable period of time.

Nevertheless, the back propagation method is one of the best known and most useful training algorithms for neural networks; it is thoroughly described in most neural network textbooks, e.g. [2]. The algorithm propagates the instantaneous squared error backward from output layer neurons through the hidden layer to the input layer during each iteration [20]. The sequence of algorithmic steps is the following. At iteration k, input data $\mathbf{x}_k$ generate the instantaneous squared error (difference between target outputs and neural network actual outputs). $\mathbf{x}_k$ passes forward from the input layer to the hidden layer, then the hidden layer signals pass to the output layer to produce the output vector $\mathbf{y}_k$. Then this actual output vector $\mathbf{y}_k$ is subtracted from the target vector $\mathbf{target}_k$ to compute the error vector $\mathbf{e}_k$. The process is repeated until it converges, that means, ideally, the algorithm has reached a local minimum of the unknown error surface.

3.1.4. Gradient descent principle in minimizing the error

Since there is number of connections within the neural network structure, the process of optimizing the performance index (such as minimizing the RMSE) consists of gradually changing

each of the connection weights in the proper direction. The process is iterative and, during each iteration, it makes all the connections slightly more efficient: each connection weight is changed in the direction leading to decreasing the error component for which that particular connection is responsible.

The back-propagation algorithm performs gradient descent on the error surface – an optimization technique for non-linear functions that attempts to incrementally move to successively lower points in search space, in order to locate a minimum [5]. The algorithm calculates the gradient vector of the error surface, or, in other words, the direction of steepest descent on the surface, and at each training epoch advances down the surface for a small distance proportional to the learning rate and the slope. All of the training algorithms use a similar strategy designed to travel towards a minimum as quickly as possible.

Because of the non-linear nature of the threshold function in most neural networks, the error surface is usually quite sophisticated. The gradient vector is directed along the path of steepest descent from the current point, thus decreasing the error when moving a small distance in that direction. A sequence of such moves will eventually find a minimum, but not necessarily the global one. The risk of being trapped in a local minimum represents a serious problem in the neural networks applications.

The size of the steps must be determined experimentally. Large steps may converge more quickly, but may also overstep the solution or (if the error surface is very eccentric) go off in the wrong direction [5]. A classic example of this is when the convergence progresses very slowly along a steep, narrow valley, bouncing from one side across to the other. In contrast, very small steps may go in the right direction, but they also require a large number of iterations. In practice, the step size is proportional to the slope (so that the objective function settles down in a minimum) and to a special constant called learning rate. The correct setting for the learning rate is application-dependent, and is usually determined empirically.

In summary, the algorithm progresses iteratively, through a number of epochs. On each epoch, the training samples are each submitted in turn to the network, and target and actual outputs compared and the error calculated. This error, together with the error surface gradient, is used to adjust the weights, and then the process repeats. The initial network configuration is random, and training stops when a given number of epochs elapses, or when the error reaches an acceptable level, or when the error stops improving.

3.1.5. Supervised versus non-supervised learning

Neural network learning algorithms can be classified into two large groups: supervised learning, one of the best known examples of which is the back-propagation algorithm, and unsupervised learning (a combination of the two can be applied as well).

In supervised learning, the network learns from a set of training data. In this approach, the user called a teacher is required to provide examples of the desired outputs for each of the input examples. The training dataset is usually taken from historical records and contains examples of inputs together with the corresponding outputs, and the network learns to infer the relationship between the two [5]. The training dataset represents information about some domain with the frequently used assumption that the features represent only properties of the examples, but not the relationships between the examples [3]. The supervised learning algorithm then searches the space of possible hypotheses for one (or more than one) solution that best estimates the unknown mapping function approximating the input-output relationships. If the network is properly trained, it has then learned to model the unknown function that relates the input variables to the output variables, and can subsequently be used to make predictions when the output is not known.

Whereas in supervised learning the training data set contains examples featuring input variables together with the associated outputs, and the network must infer a mapping from the inputs to the outputs, in unsupervised learning the training data may contain only input variables. A prime example of unsupervised learning is clustering [3]. In this case very little input, if any, is required from the user. The unsupervised learning system looks on its own the underlying structure present in the training data, through discovering the common properties in the data examples.

3.1.6. Problem of neural network over-fitting

One of the most important and desirable properties of a neural network, as well as any data analysis and prediction model, is its ability to generalize when it encounters new, unfamiliar examples of data. In reality, the network is trained to minimize the error on the training set, and short of having a perfect and infinitely large training set, this is not the same thing as minimizing the error on the real error surface - the error surface of the underlying and unknown model [5]. The most important manifestation of this distinction is the problem of over-fitting the model. This well-recognized phenomenon refers to a tendency of a learning method to force the weights to agree with the training data too closely, in order to correctly describe all of the training examples, which is done at the expense of generalization to other data, on which the trained network will be tested later [3]. The over-fitting problem is worsened by inaccuracies inevitably existing in the data, such as noise corrupted data points or inconsistent data (data points belonging to more than one category). As a result, the neural network learns in all the details not only the specificity of a particular training dataset, but also all its noise and inconsistency.

The dilemma is that a network with more complex structure and more weights models a more complex function, and is therefore prone to over-fitting, while a simpler network with less weights may not be sufficiently powerful to model the underlying function [5]. A larger and more complex network will almost invariably achieve a lower error eventually, but this may indicate over-fitting

rather than good modeling. The problems associated with local minima, and decisions over the size of network to use, imply that using a neural network typically involves experimenting with a large number of different networks, probably training each one a number of times (to avoid being fooled by local minima), and observing individual performances. The key guide to performance here is the testing error. The standard scientific percept known as Occam's razor states that the simplest explanation of the observed phenomena in a given domain is most likely to be a correct one [3]. Following the Occam's razor, a simple model is always preferable to a complex model, provided that the improvement in the testing error is negligible.

It is invariably the case that the performance of the initial network on training and testing sets is the same (if it is not at least approximately the same, the division of cases between the two sets is probably biased) [5]. As training progresses, the training error naturally drops, and providing the training process minimizes the true error function, the testing error drops too. However, if the testing error stops decreasing, or instead starts to rise, this indicates that the network is starting to over-fit the data, and training should cease. In this case, the number of hidden units and/or hidden layers should be reduced, as the network is over-powerful for the problem at hand. In contrast, if the network is not sufficiently powerful to model the underlying function, over-fitting is not likely to occur, and neither training nor testing errors will decrease to a satisfactory level. In this case, it is worthwhile to add more hidden units and/or hidden layers, to make the model adequate enough to accommodate the informational content of the dataset and be able to learn its internal structure. This must be done gradually and carefully, not at the expense of the model generalization ability.

One of the widely used techniques to fight over-fitting is early stopping, because it is simple to understand and implement and has been reported to be superior to the other methods in many cases [18]. In this technique, the training data is split into a training set and a validation set. The neural network is trained on the training set and once in while, e.g. every fifth epoch, the error is evaluated on the validation set. The training stops as soon as the error on the validation set gets higher than it was the last time it was checked. Thus, the weights the network had in that previous

epoch are stored as the result of the training. This approach uses the validation set to anticipate the behaviour on real unknown data (or on a testing set), assuming that the error on both will be similar. The validation error is used as an estimate of the generalization error. The stopping criterion is chosen to yield the lowest generalization error and also to minimize the required training time for a given generalization error. This predominantly involves a trade-off between training time and generalization error. The three classes of stopping criteria proposed in [18] are the following:

1. Stop as soon as the generalization loss exceeds a certain threshold.
2. Use the quotient of generalization loss and progress.
3. Stop when the generalization error increases in a number of successive strips.

According to [18], no class of stopping criteria has any big advantage over another (on average, for the mix of considered situations), but scaling the same criterion to be slower always tends to gain a little generalization.

3.1.7. Importance of data preprocessing in neural networks

In neural networks, it is important to scale or normalize the data fed into the network as inputs for the neurons. The transfer function of a neuron unit is typically chosen so that it can accept input in any range, and produces output in a strictly limited range (squashing effect) [5]. Although the input can be in any range, there is a saturation effect so that the unit is only sensitive to inputs within a fairly limited range. It makes sense to feed a neural network with normalized data taking on values from the interval [0,1]:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.4}$$

One of the most common transfer functions is the sigmoid function (Figure 3.1):

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.5}$$

Strictly speaking, it is only one example of sigmoid S-shaped functions, however, any other non-linear transfer function needs to resemble this shape, to be able to perform the job. In this case, the output is in the range [0,1], and the input is sensitive in a range not much larger than [-1,+1]. The function is also smooth and differentiable (Figure 3.2), facts that are critical in the neural network training algorithms. The limited numeric response range, together with the fact that information has to be in numeric form, implies that neural network solutions require pre-processing and post-processing stages to be used in real applications [5].

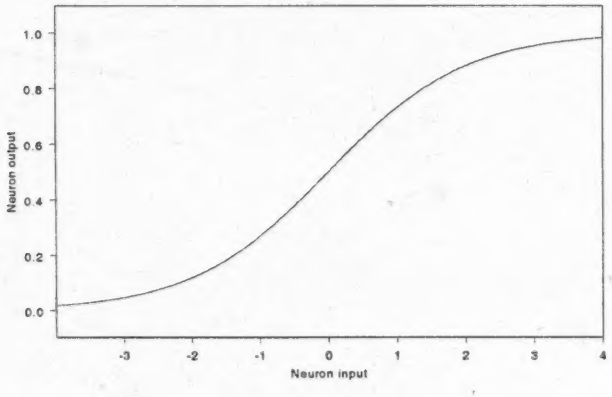


Figure 3.1 Sigmoid transfer function

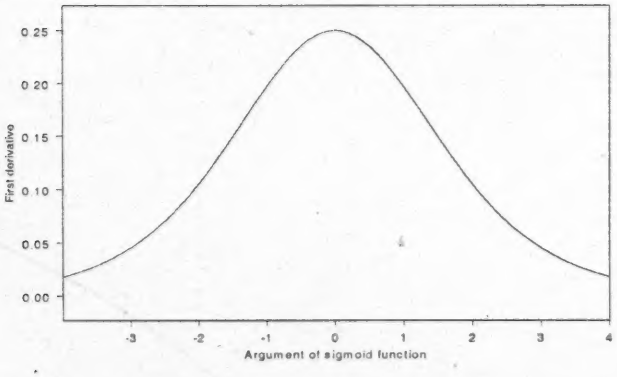


Figure 3.2 First derivative of the sigmoid function

3.2. Theoretical base of the model

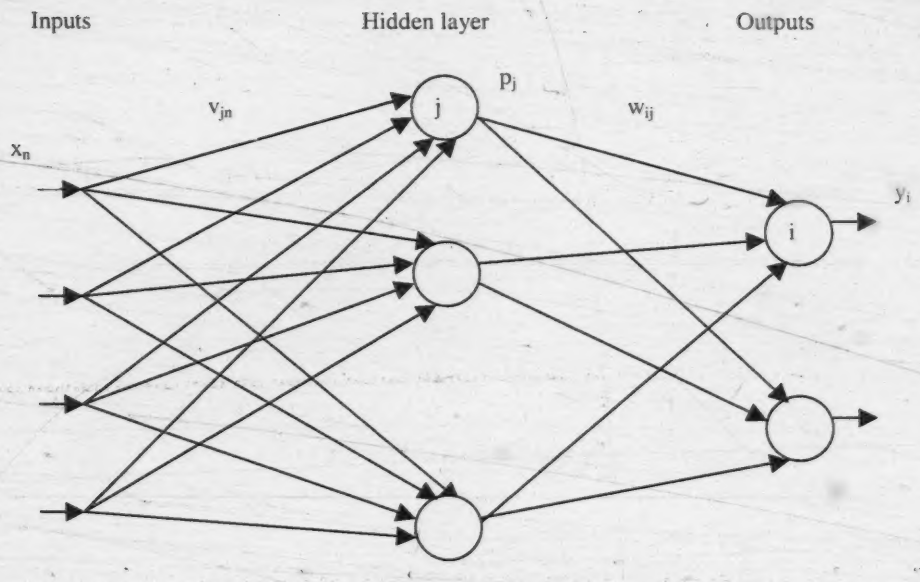
The section discusses in detail the theoretical principles behind building a feed-forward neural network with one or several hidden layers. Considered issues are: the network topology, types of transfer functions, gradient descent algorithm including computing partial derivatives, role of learning rate and momentum in training process convergence.

3.2.1. Feed-forward neural networks with one hidden layer

This is the most commonly used network architecture, due originally to Rumelhart and McClelland [21], discussed at length in most neural network textbooks, like [2]. The hidden and output layer units each perform a biased weighted sum of their inputs and pass this activation level through a transfer function to produce their output, and the units are arranged in a layered feed-forward topology. The network thus has a simple interpretation as a type of input-output model, with the weights and thresholds as free parameters of the model [5]. Such networks are theoretically proven to be universal approximators: they can model functions of arbitrary complexity, with the number of layers and the number of units in each layer determining the model complexity [2].

3.2.1.1. Topology of a neural network with one hidden layer

The feed-forward neural network structure with one hidden layer is shown in Figure 3.3. This neural network is formed in three layers, the input layer, hidden layer, and output layer, each consisting of one or more units, with full interconnection between the adjacent layers. In this particular type of neural network, the information only flows from the input to the output. Recurrent types of neural networks have more intricate connections, such as feedback paths.



x_n – one of the inputs of the neural network

p_j – output of one of the neurons (neuron j) of the hidden layer

y_i – output of one of the neurons (neuron i) of the output layer (neural network output i)

v_{jn} – connection weight between input x_n of the network and neuron j of the hidden layer

w_{ij} – connection weight between neuron j of the hidden layer and output i of the network

Figure 3.3 Topology of a feed-forward neural network with one hidden layer

The nodes of the input layer are passive, meaning they do not modify the data [4]. They receive a single value on their input, and duplicate the value to their multiple outputs. In comparison, the nodes of the hidden and output layer are active, they modify the data passing them through their transfer function. Each value from the input layer is duplicated and sent to all hidden units, thus creating a fully interconnected structure. As shown in Figure 3.3, the values entering a hidden node are multiplied by weights, a set of predetermined coefficients corresponding to each connection that are stored in the model. The weighted inputs are then added together to produce a

single number. Before leaving the unit, this number is passed through a nonlinear transfer function of sigmoid shape that limits the unit's output into a specified interval. The outputs from the hidden layer are in turn duplicated and applied to the next layer. The active units of the output layer combine and modify the data from previous layer to produce the network output values.

The neural network is more flexible when the transfer function could be adjusted to the left or right from x=0, making it centered on some other value [4]. To implement this, an additional unit is added to the input layer, with its input always having a value of 1. When this value is multiplied by the weights of the hidden layer, it adds an offset to each sigmoid. This addition is called a bias. It is treated the same as the other inputs, except that it has a constant value.

3.2.1.2. Number of neurons in hidden layer

Important issues in the feed-forward neural network design include specification of the number of hidden layers and the number of units in these layers. The number of input and output units is determined by the problem. There may be some uncertainty about precisely which inputs to use, however, it can be assumed that when the input variables are intuitively selected, they are all meaningful. The number of hidden units to use is far from clear. As good starting point as any is to use one hidden layer, with the number of units equal to half the sum of the number of input and output units [5].

Once the input variables have been selected, the network design follows a number of stages [5]:

1. Select an initial configuration (typically, one hidden layer with the number of hidden units set to half the sum of the number of input and output units).
2. Iteratively conduct a number of experiments with each configuration, retaining the best network (in terms of testing error) found. A number of experiments are required with each

configuration to avoid being fooled if the training process locates a local minimum, and it is also best to resample.

3. During the course of experimentation, if under-learning occurs (the network does not achieve an acceptable performance level) try adding more neurons to the hidden layer(s). If this does not help, try adding an extra hidden layer.
4. If over-fitting occurs (selection error starts to rise), try removing hidden units (and possibly layers).
5. Once an effective network configuration has been experimentally determined, resample and generate new networks with that configuration.

3.2.1.3. Types of transfer functions

Nonlinear activation functions are what give neural networks their nonlinear capabilities [19]. One of the most common forms of activation function is the sigmoid (Figure 3.1) which is a monotonically increasing function that asymptotes at some finite value as $\pm\infty$ is approached. The exact shape of the sigmoid is not important, only that it provides a smooth threshold [4]. For comparison, a simple step threshold function called a hard-limiter produces a value of one when $x>0$, and a value of zero when $x<0$. However, without a sigmoid or similar non-linearity, the summations and weights of the hidden and output layers could be combined into a single layer. This would result in a primitive two-layer linear network that would be very limited in its approximation capabilities.

The sigmoid (3.5) performs the same basic thresholding function as the simple step hard-limiter, but, besides that, it is differentiable, as shown in Figure 3.2. The derivative is a critical part of finding the proper weights in the gradient descent technique. For the sigmoid (3.5), there is a shortcut to calculating the value of its first derivative, to make the algebra shorter [4]:

$$f'(x) = f(x)(1 - f(x)) \tag{3.6}$$

Some of the other examples of nonlinear differentiable transfer functions having similar shape to that shown in Figure 3.1 are the arctangent (Figure 3.4) and hyperbolic tangent (Figure 3.5) functions, with the expressions for their first derivatives:

$$f(x) = \frac{\text{arctg}(x)}{\pi} + 0.5 \qquad f'(x) = \frac{1}{\pi(1 + x^2)} \tag{3.7}$$

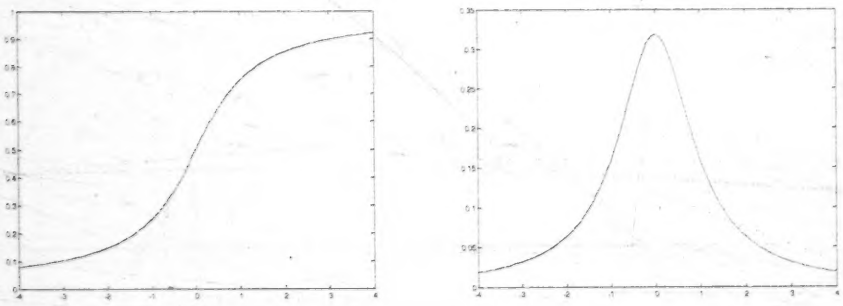


Figure 3.4 Plot of arctangent function and its first derivative

$$f(x) = \frac{\tanh(x) + 1}{2} \qquad f'(x) = \frac{1 - x^2}{2} \tag{3.8}$$

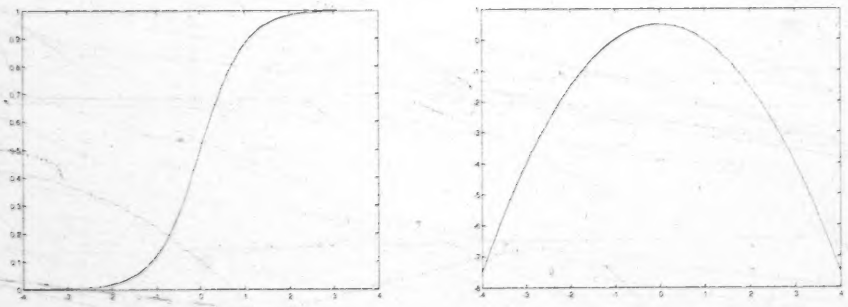


Figure 3.5 Plot of hyperbolic tangent function and its first derivative

3.2.1.4. Implementation of the gradient descent algorithm

The back-propagation algorithm works by performing gradient descent on the error surface. Speaking loosely, it calculates the direction of steepest descent on the surface, and jumps down the surface a distance proportional to the learning rate and the slope, picking up momentum as it maintains a consistent direction [5]. The descent is calculated independently on the error surface for each training case, and in random order, but this is actually a good approximation to descent on the composite error surface.

Once the network architecture is defined, the only parameters that can be changed in the process of the neural network training are the connection weights between the layers. The procedure of iterative adjusting the weights in such a way that the network's error (MSE or RMSE) is minimized is the heart of the neural network learning strategy [4]. In general, the gradient descent on a surface is performed by moving along each dimension of the surface for a distance proportional to the slope along that dimension. For the neural network error surface, the dimensions are represented by the parameters modified by the training algorithm, i.e. the interlayer connection weights. Thus, the gradient descent on the error surface is performed through calculating the slope for each weight, and then changing each weight by an amount proportional to that slope. The slope of each neural network weight is calculated as a network output's partial derivative with respect to this particular weight: by adding a small increment to the weight value ∂w, finding the resulting change in the output signal ∂y, and dividing the two: $\partial y/\partial w$.

That is why the sigmoid needs to be differentiable [4]. If the slope is known at each point of the sigmoid, an equation can be written for the slope of each weight $\partial y/\partial w$ of the network in the Figure 3.3. Considering a specific weight, for example, w_{ij}, corresponding to the connection between hidden layer neuron j (while its output p_j stays the same) and the output layer neuron i producing the output y_i, the partial derivative defines the way how the output y_i can be affected by a small change of this particular weight w_{ij}, while everything else is kept the same [4]:

$$\frac{\partial y_i}{\partial w_{ij}} = y_i' p_j \tag{3.9}$$

where y_i' is the first derivative of the output layer sigmoid, describing how much the output of the sigmoid of the unit y_i changes in response to a small change in the input to the sigmoid. According to (3.6), y_i' can be calculated from the current output value y_i of the sigmoid:

$$y_i' = y_i(1 - y_i) \tag{3.10}$$

Thus, the partial derivative for the output layer weight w_{ij} is computed as follows:

$$\frac{\partial y_i}{\partial w_{ij}} = y_i(1 - y_i)p_j \tag{3.11}$$

Using a similar analysis, the slope for a weight on the hidden layer, such as v_{jn}, can be found as a product of two partial derivatives. The first of them reflects the change of the output layer sigmoid y_i in response to the change of the hidden layer neuron output p_j, with the fixed weight w_{ij} of the connection them. The second factor reflects the change of the hidden layer sigmoid p_j in response to the change of the weight v_{jn} of the connection between the network input x_n and the neuron j of the hidden layer, with the fixed value of the input x_n:

$$\frac{\partial y_i}{\partial v_{jn}} = \frac{\partial y_i}{\partial p_j} \cdot \frac{\partial p_j}{\partial v_{jn}} = \left(y_i(1 - y_i)w_{ij}\right) \cdot \left(p_j(1 - p_j)x_n\right) \tag{3.12}$$

where $y_i(1 - y_i)$ is the first derivative of the output layer sigmoid, and $p_j(1 - p_j)$ is the first derivative of the hidden layer sigmoid, evaluated where we are operating on their curves. The

other values, x_n and w_{ij}, are simply constants that the weight change sees as it makes its way to the output [4].

When the slope of each of the weights is found, the gradient descent rule specifies how each weight is changed for the next iteration. The new value for each particular output layer weight w_{ij} is found by taking the current weight and adding delta – an adjustment amount that is proportional to the slope [4]:

$$w_{ij}(iter+1) = w_{ij}(iter) + \Delta w_{ij} \tag{3.13}$$

where

$$\Delta w_{ij} = \alpha \cdot (target_i - y_i) \cdot \frac{\partial y_i}{\partial w_{ij}} \tag{3.14}$$

Similarly, for the hidden layer weights:

$$v_{jn}(iter+1) = v_{jn}(iter) + \Delta v_{jn} \tag{3.15}$$

where

$$\Delta v_{jn} = \alpha \cdot (target_i - y_i) \cdot \frac{\partial y_i}{\partial v_{jn}} \tag{3.16}$$

The proportionality constant consists of two factors: $(target_i - y_i)$, which is the error on the output i of the neural network for this particular data input, and α, a learning rate constant defined at the beginning of the algorithm. The role of the error term in this calculation is intuitively clear: when y_i is less than $target_i$, each weight needs to be increased, and the bigger the difference, the bigger increase should be. And vice versa, when y_i exceeds the $target_i$, each weight needs to be decreased by the value proportional to the difference.

3.2.1.5. Influence of learning rate

The effectiveness and convergence of the back-propagation learning algorithm depends significantly on the value of the learning rate. The learning rate controls how much the weights are changed at each iteration. In general, the optimum value depends on the problem being solved, and there is no single value suitable for different training cases. The value to use depends on the particular problem, being as low as 0.000001, or as high as 0.1, according to [4]. Obviously, too small value will cause the network to converge too slowly. In contrast, too large value will cause the convergence to be erratic, exhibiting chaotic oscillation around the final solution, and may even never stabilize at any minimum. The way neural networks react to various values of the learning rate can be difficult to understand or predict. It should be determined experimentally, and can be modified in the process of training. This makes it critical that the network performance index (accumulated error such as MSE or RMSE) is monitored during the training, at the end of each iteration. If the system is not converging properly, the algorithm needs to try another value for the learning rate [4].

3.2.1.6. Momentum in learning

The algorithm is also often modified by inclusion of a momentum term: this encourages movement in a fixed direction, so that if several steps are taken in the same direction, the algorithm gains the momentum, which gives it the ability to escape local minimum (although not necessarily), and also to move rapidly over flat spots and plateaus [5].

The purpose of momentum term is therefore to accelerate the convergence of the back-propagation learning algorithm. Speed of learning is governed by learning rate, but if the learning rate is too high, the process becomes unstable, oscillating back and forth across the error minimum. Adding a momentum term that includes a proportion of the last weight change is one of the possible ways to

overcome the speed limitation imposed by the learning rate that needs to be low enough to ensure stable and consistent learning:

$$\Delta w_{ij}(iter+1) = \Delta w_{ij}(iter) + \mu \cdot \Delta w_{ij}(iter) = (1+\mu) \cdot \Delta w_{ij}(iter) \tag{3.17}$$

And for the hidden layer weights:

$$\Delta v_{jn}(iter+1) = \Delta v_{jn}(iter) + \mu \cdot \Delta v_{jn}(iter) = (1+\mu) \cdot \Delta v_{jn}(iter) \tag{3.18}$$

This way, if the previous weight change was large, the new change will be even larger. In other words, the weight change carries along some momentum to the next iteration. This has a tendency to smooth out small fluctuations in the error-weight space (low pass filter effect). And, similarly to the learning rate, if the momentum is too high, it can result in over-jumping the minima and even never stabilizing. Typically, the momentum term is chosen between 0.1 and 0.9, and, depending on this, the number of epochs needed to train the network may differ by an order of magnitude for different values of the momentum.

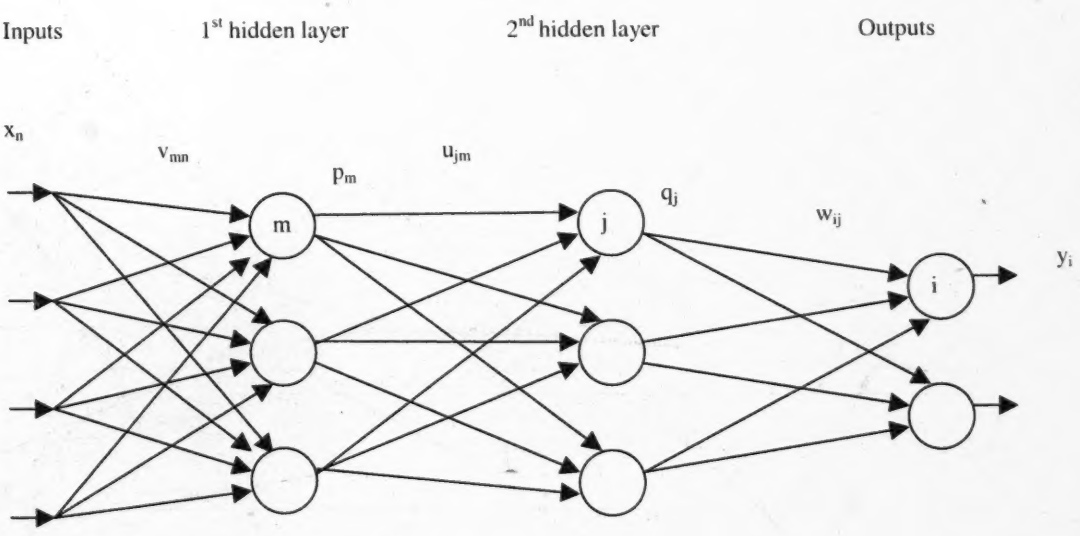
3.2.2. Neural network with several hidden layers

This subsection discusses the reasons of introducing additional layers of neurons and principles of multi-layer neural network organization, considering in particular two-layer network topology and focusing on the implications of computing partial derivatives for two-layer network.

3.2.2.1. Topology of multi-layer network

There is a trade-off between learning precision and generalization ability of a neural network, which is regulated by the complexity of the network. Occam's razor suggests choosing the

simplest possible configuration able to deliver acceptable results, as a starting point. If the performance index improves significantly on the testing dataset when increasing the network complexity, it makes sense to increase it. The first obvious step for increasing the complexity of a simple fully connected feed-forward network with one hidden layer is increasing the number of neurons in the hidden layer. If after a certain point this starts to bring negligible (if any at all) improvements in testing performance index, the next step would be to try to introduce a second hidden layer, also fully interconnected with the first hidden layer and with the output layer (Figure 3.6).



x_n – one of the inputs of the neural network

p_m – output of one of the neurons (neuron m) of the first hidden layer

q_j – output of one of the neurons (neuron j) of the second hidden layer

y_i – output of one of the neurons (neuron i) of the output layer (network output i)

v_{mn} – connection weight between input x_n of the network and neuron m of the first hidden layer

u_{jm} – connection weight between neuron m of the first hidden layer and neuron j of the second hidden layer

w_{ij} – connection weight between neuron j of the second hidden layer and neuron i of the output layer (neural network output i)

Figure 3.6 Topology of a feed-forward neural network with two hidden layers

3.2.2.2. Computing partial derivatives

An additional hidden layer makes no influence on computing the partial derivatives with respect to the output layer weights w_{ij}. Thus, the formula (3.11) can be applied without changes:

$$\frac{\partial y_i}{\partial w_{ij}} = y_i(1-y_i)q_j \tag{3.11}$$

For the partial derivatives with respect to the weights u_{jm} of connections between first and second hidden layers, the analogy of the formula (3.12) is used. The modification is that the constant x_n corresponding to the value of the network input x_n is substituted by the value p_m produced by the sigmoid of the neuron m of the first hidden layer:

$$\frac{\partial y_i}{\partial u_{jm}} = \frac{\partial y_i}{\partial q_j} \cdot \frac{\partial q_j}{\partial u_{jm}} = y_i(1-y_i)w_{ij}q_j(1-q_j)p_m \tag{3.19}$$

Computing the partial derivatives with respect to the weights v_{mn} of connections between the neural network inputs and the first hidden layers is more complicated. Each of these weights can affect all j neurons of the second hidden layers, thus making influence on the neural network output y_i through j possible paths. Each of those j paths can be represented as a product of three factors:

1. the change of the output layer sigmoid y_i in response to the change of the second hidden layer neuron output q_j, with the fixed weight w_{ij} of the connection them;
2. the change of the second hidden layer sigmoid q_j in response to the change of the first hidden layer sigmoid qp_j with the fixed weight w_{ij} of the connection them;
3. the change of the first hidden layer sigmoid p_m in response to the change of the weight v_{mn} of the connection between the network input x_n and the neuron m of the first hidden layer, with the fixed value of the input x_n.

There must be summation by j in the formula, to consider the influence of all j paths:

$$\frac{\partial y_i}{\partial v_{mn}} = \sum_j \left(\frac{\partial y_i}{\partial q_j} \cdot \frac{\partial q_j}{\partial p_m} \cdot \frac{\partial p_m}{\partial v_{mn}} \right) = \sum_j \left(y_i (1 - y_i) w_{ij} q_j (1 - q_j) u_{jm} p_m (1 - p_m) x_n \right) \tag{3.20}$$

3.2.3. Neural network with feedback from the target outputs

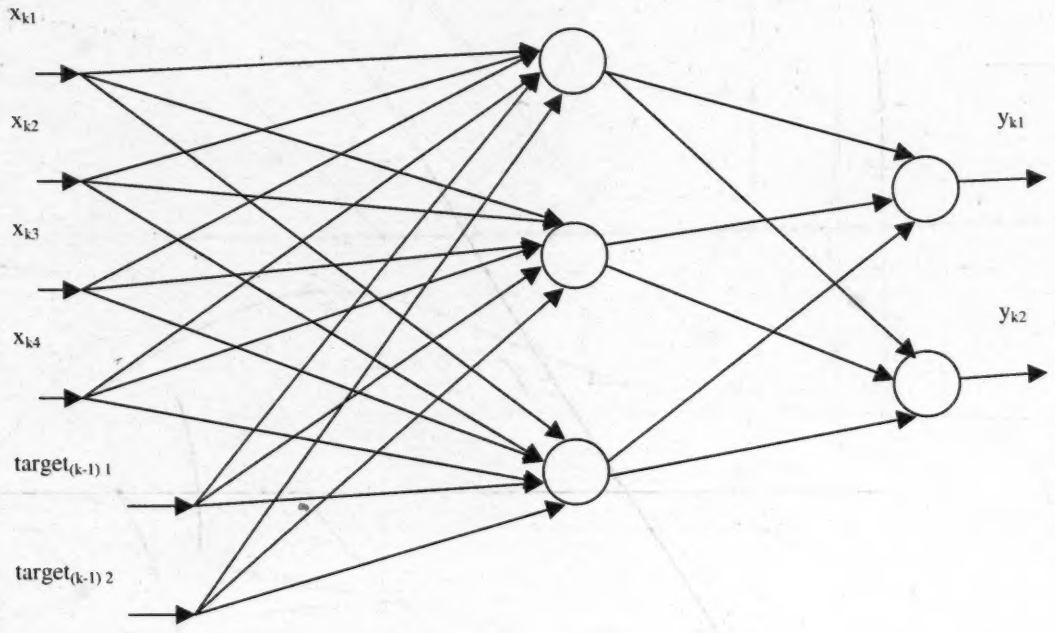
This subsection introduces the notion of dynamic neural network model looking additionally at the data points belonging to previous moments of time, as opposed to the static models analyzing only current moment data. The hypothesis is that a neural network will perform more precisely the analysis and prediction of the data in presence of the available information about the target outputs from one or several previous data points. These target outputs are fed into the neural network as additional inputs, thus creating feedback.

3.2.3.1. Rationale behind introducing a feedback

The data collected for analysis often represent a time series, when each data point is registered at a certain moment of time, one after another. Classical example is macroeconomic forecasting, where the most reliable and popular models have been Bayesian vector autoregressive models. However, Moody [22] demonstrated that, relative to conventional linear time series and regression methods, superior performance can be obtained using state-of-the-art neural network models. He found that neural networks often outperform standard linear time series models.

Selecting a best subset of input variables is a critical part of model selection for forecasting [22]. In case of time series, the dataflow fed into the neural network may contain some regularity in terms of time. While in a static neural network model the output of the network is simply a function of its inputs, in a dynamic model operating with time series, the output can be considered

not only a function of inputs, but also a function of time. In this sense, the target outputs corresponding to one or more previous data points may be used as extra arguments, in addition to the existing inputs (Figure 3.7), for the model trying to approximate and predict outputs as a function of inputs. The hypothesis is that this additional information may be useful for the neural network in the process of learning from historically collected data which may result in more accurate connection weights providing better performance on the testing dataset.



$x_{k1}, \ldots, x_{k4}$ – neural network inputs corresponding to the data point k

y_{k1}, y_{k1} – neural network outputs corresponding to the data point k

$target_{(k-1)\,1}, target_{(k-1)\,2}$ – target outputs corresponding to the data point k–1

Figure 3.7 Topology of a neural network with feedback from the target outputs

3.2.3.2. Method of feedback implementation

The feedback from the previous cycle target outputs can be implemented in the form of the extra columns added to the original dataset to be considered as extra inputs of the neural network. These extra input columns are formed from shifted output columns of the original dataset. For example, if the analyzed dataset looks as follows:

Input 1	Input 2	Input 3	Input 4	Output 1	Output 2
x_{11}	x_{12}	x_{13}	x_{14}	$target_{11}$	$target_{12}$
x_{21}	x_{22}	x_{23}	x_{24}	$target_{21}$	$target_{22}$
x_{31}	x_{32}	x_{33}	x_{34}	$target_{31}$	$target_{32}$
...	...	...	...	...	...
x_{k1}	x_{k2}	x_{k3}	x_{k4}	$target_{k1}$	$target_{k2}$

then modified dataset with the feedback from the previous cycle target outputs would be:

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Output 1	Output 2
x_{11}	x_{12}	x_{13}	x_{14}	0	0	$target_{11}$	$target_{12}$
x_{21}	x_{22}	x_{23}	x_{24}	$target_{11}$	$target_{12}$	$target_{21}$	$target_{22}$
x_{31}	x_{32}	x_{33}	x_{34}	$target_{21}$	$target_{22}$	$target_{31}$	$target_{32}$
...	...	...	...	...	...	...	...
x_{k1}	x_{k2}	x_{k3}	x_{k4}	$target_{(k-1)1}$	$target_{(k-1)2}$	$target_{k1}$	$target_{k2}$

with the feedback from two previous cycles:

Inp 1	Inp 2	Inp 3	Inp 4	Input 5	Input 6	Input 7	Input 8	Output 1	Output 2
x_{11}	x_{12}	x_{13}	x_{14}	0	0	0	0	$target_{11}$	$target_{12}$
x_{21}	x_{22}	x_{23}	x_{24}	$target_{11}$	$target_{12}$	0	0	$target_{21}$	$target_{22}$
x_{31}	x_{32}	x_{33}	x_{34}	$target_{21}$	$target_{22}$	$target_{11}$	$target_{12}$	$target_{31}$	$target_{32}$
...	...	...	...	...	...	...	...	...	...
x_{k1}	x_{k2}	x_{k3}	x_{k4}	$target_{(k-1)1}$	$target_{(k-1)2}$	$target_{(k-2)1}$	$target_{(k-2)2}$	$target_{k1}$	$target_{k2}$

and so on.

3.3. Feed-forward neural network model applied to the analyzed dataset

First, the experiments were conducted with a feed-forward neural network with one hidden layer. The architecture described in Figure 3.3 was implemented through the software model. The experiments were organized as follows:

1. The dataset was normalized along each dimension (input and output) into the interval [0,1], using linear normalization (scaling) formula (3.3).
2. The first 2,155 data points (68%) were selected for the training dataset, to be used as examples for training the connections of the neural network. The last 1,015 data points (32%) served as a testing dataset to validate the model.
3. All of the neural network connections were initiated as random values in the interval [0,1].
4. The network was trained during the specified number of iterations, on the training dataset, in the following way:

   Compute the RMSE performance index (3.3) for the network with initial random weights. Then, for each training epoch (iteration):

   - Feed the network with the input values of every data point
   - Compute the network outputs for every data point
   - Compute the errors between target outputs and network actual outputs
   - Compute the output partial derivatives with respect to the output layer weights according to (3.11)
   - Compute the output partial derivatives with respect to the hidden layer weights according to (3.12)
   - Compute the new weights according to (3.13) – (3.16) and update the network
   - Compute the performance index for the network with updated weights

5. The trained network was tested on the testing dataset by computing the RMSE performance index (3.3) for all of the data points of the testing dataset.

In the experiments described below, the following parameters of the neural network were modified, to explore their influence on the model performance:

- number of neurons in the hidden layer
- number of learning epochs
- learning rate
- momentum
- type of neuron transfer function in the hidden layer
- linear vs. non-linear output layer transfer function
- feedback from previous cycle target outputs

Besides experimenting with the neural networks with one hidden layer, the effect of adding a second hidden layer was analyzed as well in a series of additional experiments. In these experiments, the software model was modified to implement the architecture described in Figure 3.6 and to compute the output partial derivatives according to formulas (3.11), (3.19), and (3.20).

3.3.1. Experimenting with number of learning epochs

Every experiment started from the random values of all the neural network connection weights, so in the first iteration the training error was high, but it always declined very fast in the next iteration. After the first correction, the error continued to decline gradually, converging to some value corresponding to a minimum found by the network on the error surface. Stopping an experiment after 1000 learning epochs gave an impression of a steady convergence (Figure 3.8). However, running the same experiment for 4000 epochs revealed the convergence pattern rising and declining again (Figure 3.9). It can be presumed that the discovered minimum was one of the local ones and that the model was able to overcome it and converge to another local minimum.

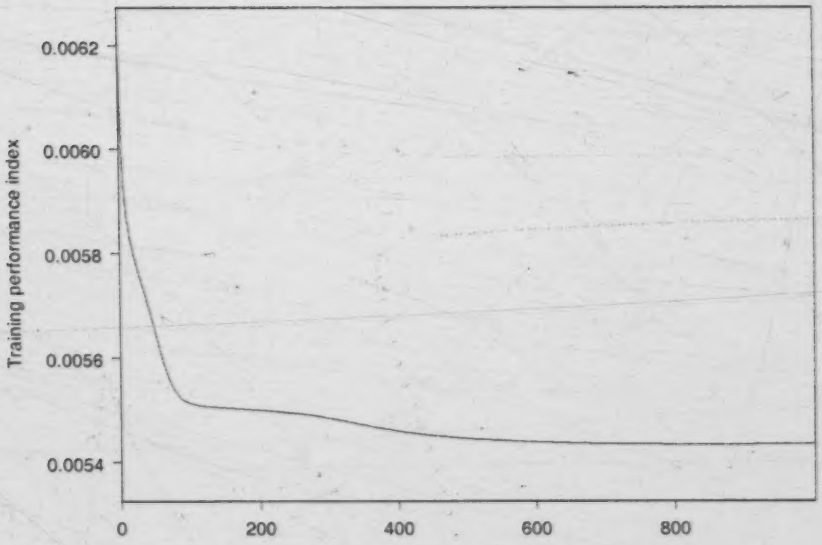


Figure 3.8 Tracing the performance index during 1000 training epochs

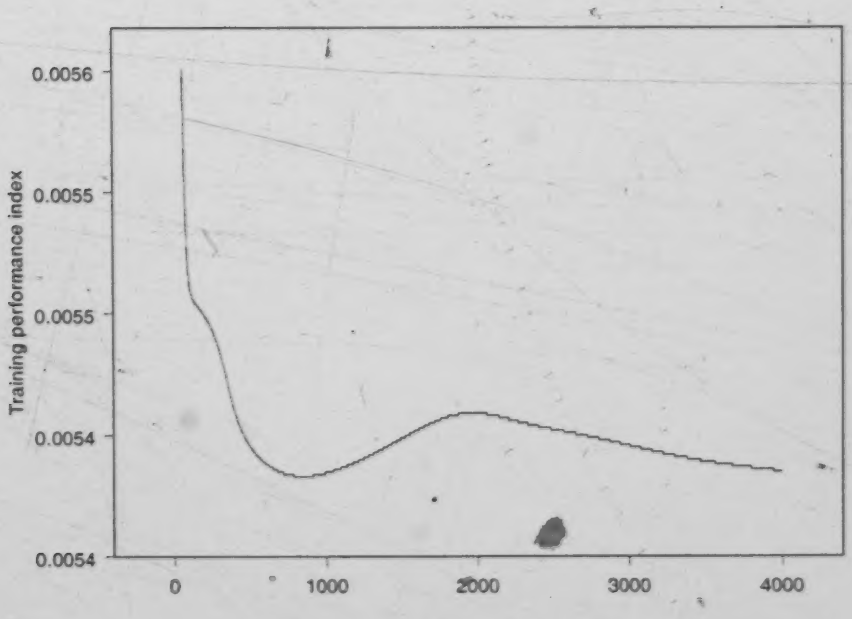


Figure 3.9 Tracing the performance index during 4000 training epochs

Extending the number of learning epochs to 10,000 demonstrated continued convergence that allowed the neural network to reach the RMSE value slightly lower than at the local minimum found around 850 iterations. However, the improvement reached with more than 4000 iterations was negligible, so the number of learning epochs was chosen to be 4000 for all the other experiments.

3.3.2. Experimenting with learning rate and with number of hidden neurons in a network with one hidden layer

The experiment was repeated for 24 different numbers of hidden neurons (from 2 to 25), and for 10 different values of learning rate (from 0.005 to 0.05, with the increment of 0.005). This way, the results were obtained for 240 different models (Table 3.1). The parameters were as follows: number of learning epochs fixed to be 4000, no momentum term, sigmoid transfer function in the neurons of both hidden and output layers. On the testing dataset, the best performance (RMSE = 0.005680) was obtained by the network with 5 neurons in the hidden layer, trained with the learning rate of 0.015 (marked with bold face in both tables and with an arrow in Figure 3.13).

For this configuration, some of the model outputs (y1 – y3) are presented (Figure 3.10, Figure 3.11, and Figure 3.12) vs. targets for the training and testing data. The plots for the network performance on the outputs y4, y5, and y6 resemble respective plots for the outputs y1, y2, and y3, so they are not presented on the figures. The plots are built on the fragments of the 4th operational cycle of the training dataset and of the 1st operational cycle of the testing dataset.

Table 3.1 Performance index RMSE for each experiment with static neural network with one hidden layer, all sigmoid transfer functions, no momentum, no feedback.

For training data:

	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
2	0.005435	0.005452	0.005472	0.00549	0.005497	0.005497	0.005505	0.005572	0.005520	0.005676
3	0.005376	0.005440	0.005462	0.005437	0.005452	0.005448	0.005450	0.005454	0.005480	0.005483
4	0.005387	0.005421	0.005406	0.005427	0.005465	0.005440	0.005417	0.005440	0.005492	0.005480
5	0.005376	0.005410	**0.005411**	0.005445	0.005383	0.005366	0.005418	0.005383	0.005482	0.005431
6	0.005387	0.005376	0.005394	0.005374	0.005416	0.005446	0.005299	0.005362	0.005343	0.005394
7	0.005367	0.005335	0.005332	0.005337	0.005294	0.005319	0.005377	0.005332	0.005315	0.005303
8	0.005351	0.005299	0.005321	0.005294	0.005314	0.005297	0.005344	0.005334	0.005336	0.005351
9	0.005357	0.005343	0.005337	0.005325	0.005276	0.005328	0.005268	0.005346	0.005370	0.005279
10	0.005331	0.005336	0.005308	0.005302	0.005301	0.005329	0.005250	0.005335	0.005288	0.005318
11	0.005378	0.005306	0.005310	0.005288	0.005304	0.005294	0.005271	0.005283	0.005299	0.005265
12	0.005337	0.005298	0.005277	0.005261	0.005251	0.005253	0.005291	0.005271	0.005286	0.005300
13	0.005330	0.005303	0.005304	0.005277	0.005272	0.005233	0.005240	0.005286	0.005246	0.005305
14	0.005342	0.005330	0.005303	0.005257	0.005253	0.005252	0.005293	0.005249	0.005218	0.005307
15	0.005332	0.005336	0.005292	0.005288	0.005248	0.005249	0.005452	0.005283	0.005284	0.005257
16	0.005328	0.005317	0.005240	0.005303	0.005232	0.005238	0.005307	0.005220	0.005232	0.005278
17	0.005342	0.005320	0.005318	0.005245	0.005334	0.005262	0.005217	0.005229	0.005288	0.005260
18	0.005308	0.005313	0.005286	0.005291	0.005229	0.005321	0.005244	0.005252	0.005273	0.005301
19	0.005314	0.005342	0.005266	0.005246	0.005265	0.005259	0.005271	0.005257	0.005246	0.005308
20	0.005351	0.005304	0.005287	0.005218	0.005283	0.005275	0.005265	0.005238	0.005296	0.005229
21	0.005338	0.005269	0.005285	0.005231	0.005285	0.005279	0.005250	0.005294	0.005278	0.005350
22	0.005349	0.005281	0.005268	0.005288	0.005220	0.005230	0.005182	0.005234	0.005200	0.005208
23	0.005335	0.005287	0.005279	0.005228	0.005223	0.005239	0.005246	0.005270	0.005263	0.005198
24	0.005330	0.005294	0.005254	0.005259	0.005236	0.005241	0.005216	0.005207	0.005334	0.005219
25	0.005342	0.005296	0.005255	0.005249	0.005268	0.005233	0.005228	0.005219	0.005232	0.005252

For testing data:

	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
2	0.005829	0.005763	0.005747	0.005747	0.005744	0.005764	0.005762	0.005798	0.005789	0.005852
3	0.005775	0.005743	0.005763	0.005708	0.005701	0.005703	0.005711	0.005717	0.005697	0.005725
4	0.005805	0.005740	0.005709	0.005704	0.005691	0.005706	0.005724	0.005716	0.005717	0.005710
5	0.005762	0.005698	**0.005680**	0.005682	0.005830	0.005797	0.005735	0.005827	0.005747	0.005745
6	0.005761	0.005766	0.005743	0.005818	0.005859	0.005849	0.005845	0.005767	0.005749	0.005765
7	0.005820	0.005832	0.005882	0.005780	0.005869	0.005845	0.005864	0.005937	0.005776	0.005951
8	0.005751	0.005835	0.005778	0.005846	0.005796	0.005811	0.005877	0.005885	0.005851	0.005772
9	0.005773	0.005798	0.005807	0.005913	0.005898	0.005909	0.005838	0.005970	0.005896	0.005947
10	0.005762	0.005884	0.005810	0.005903	0.005924	0.005951	0.005822	0.005919	0.005911	0.005947
11	0.005770	0.005818	0.005815	0.005918	0.005895	0.005898	0.005931	0.005912	0.005897	0.006025
12	0.005842	0.005825	0.005868	0.005891	0.005917	0.005988	0.005986	0.006015	0.005898	0.005974
13	0.005821	0.005812	0.005874	0.005791	0.005878	0.005855	0.005877	0.005903	0.005961	0.005970
14	0.005796	0.005794	0.005787	0.005810	0.005824	0.005923	0.005920	0.005976	0.006052	0.006093
15	0.005790	0.005774	0.005896	0.005782	0.005889	0.006031	0.006052	0.006031	0.006030	0.006001
16	0.005815	0.005777	0.005842	0.005877	0.005912	0.006013	0.005868	0.005938	0.005949	0.005960
17	0.005764	0.005853	0.005923	0.005891	0.005931	0.005924	0.005951	0.005928	0.005978	0.005918
18	0.005842	0.005775	0.005886	0.005858	0.005881	0.005984	0.005935	0.005950	0.005962	0.006003
19	0.005849	0.005800	0.005831	0.005851	0.005859	0.005877	0.005965	0.005981	0.005951	0.005975
20	0.005797	0.005872	0.005865	0.005926	0.005836	0.005976	0.005922	0.005928	0.006000	0.005897
21	0.005803	0.005815	0.005879	0.005909	0.005857	0.005915	0.006011	0.005925	0.005970	0.005940
22	0.005788	0.005864	0.005797	0.005854	0.005854	0.005853	0.005940	0.005926	0.005983	0.006017
23	0.005820	0.005803	0.005853	0.005866	0.005904	0.005910	0.005868	0.006022	0.005988	0.005948
24	0.005842	0.005830	0.005862	0.005850	0.005870	0.005947	0.005957	0.005920	0.005913	0.005954
25	0.005796	0.005868	0.005825	0.005888	0.005864	0.005914	0.005941	0.006047	0.006004	0.005873

First column denotes the number of neurons in the hidden layer, first row – learning rate.

Upper table represents the results for training dataset, lower table – for testing dataset.

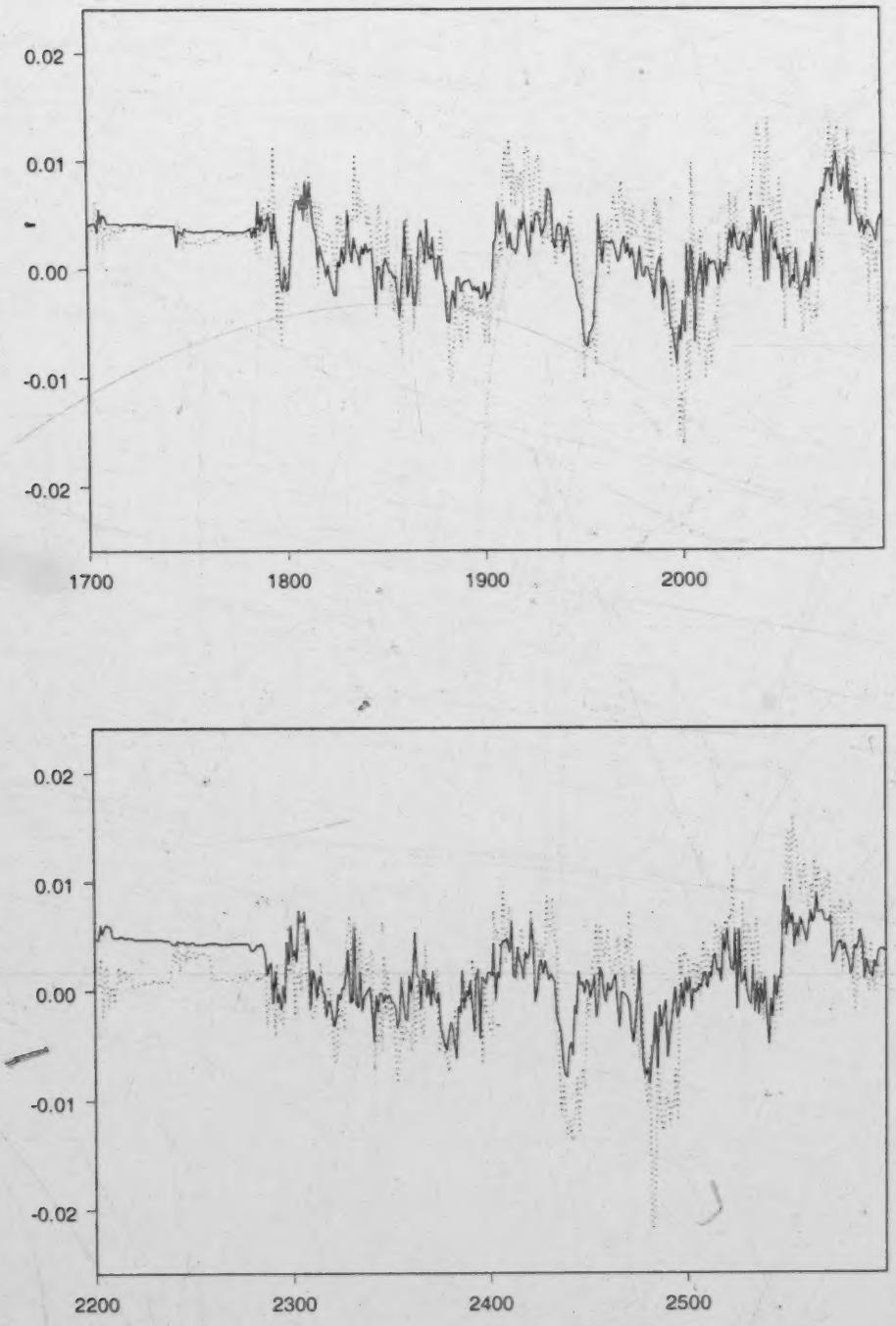


Figure 3.10 Output y1 (Floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training dataset, lower plot – fragment of testing dataset.

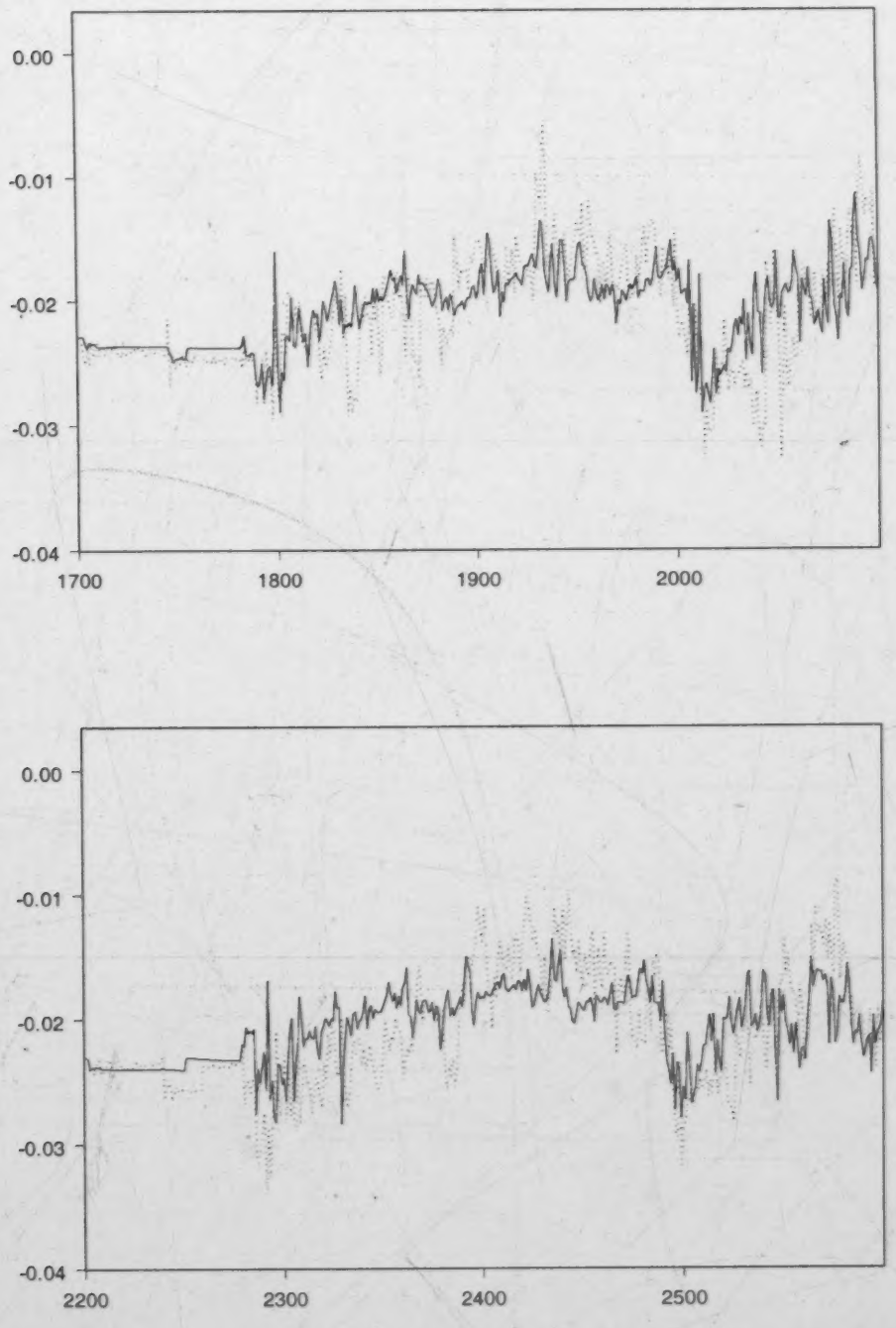


Figure 3.11 Output y2 (Floor Y acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training dataset, lower plot – fragment of testing dataset.

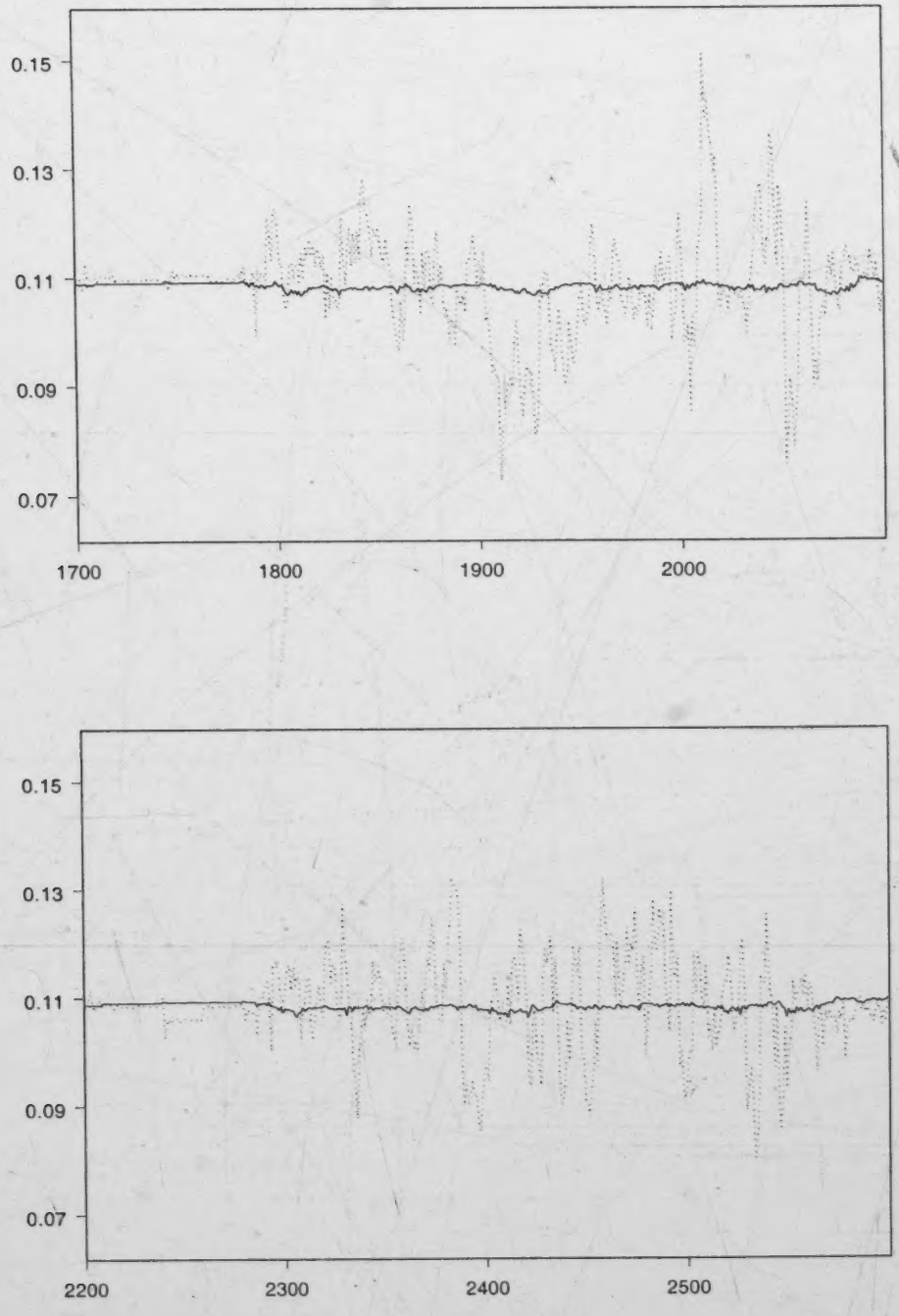


Figure 3.12 Output y3 (Floor Z acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training dataset, lower plot – fragment of testing dataset.

We can see that the network models the target outputs y1, y2, y4, and y5 quite reasonably, but totally fails on the outputs y3 and y6, which are the vertical (Z) components of measured accelerations, thus the most important ones to be modeled. Possible explanation could be that for the outputs y1, y2, y4, and y5 (horizontal acceleration components), their values and standard deviations are very similar, while for the outputs y3 and y6 (vertical components), the amplitude of the values is an order of magnitude as higher, and standard deviation is twice as higher than for the other 4 outputs. A neural network, in the process of training, adjusts its weights to minimize the error on the maximum possible number of outputs. In other words, it adjusts itself to those four outputs that differ very little from each other in amplitude and standard deviation, while ignoring the other two outputs that are different from those four.

Figure 3.13 presents visualization of the network performance index RMSE as a function of number of neurons in the hidden layer and learning rate, and demonstrates very well the effect of the network over-fitting. While on the training dataset the error visibly decreases with an increase of the number of neurons in the hidden layer, we can see that on the testing dataset the best results are obtained for the simplest configurations. With increase of the number of neurons in the hidden layer, the network learns too closely all the peculiarities of the training dataset and looses the ability to generalize the learnt knowledge when presented with the testing dataset.

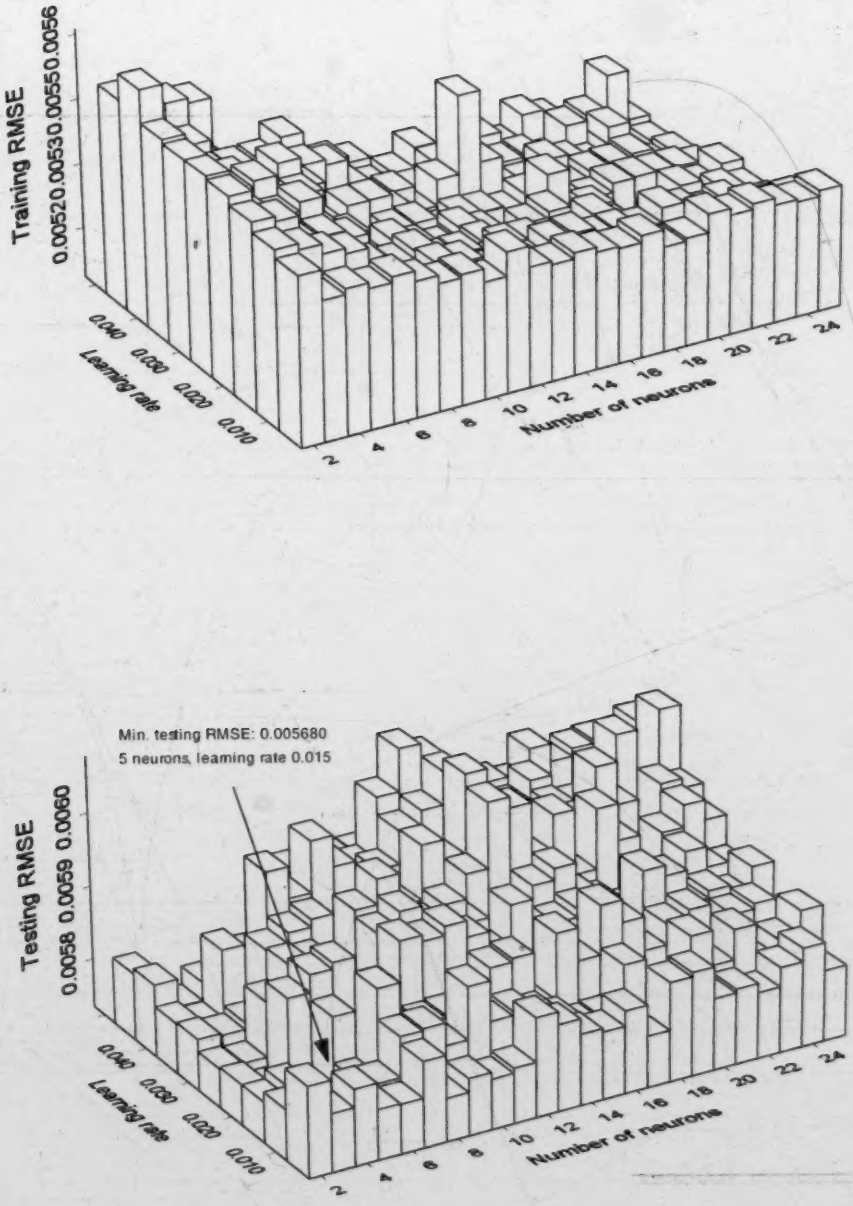


Figure 3.13 RMSE as a function of number of neurons in the hidden layer and learning rate. Upper plot for training dataset, lower plot – for testing dataset. Output layer sigmoid.

3.3.3. Cross-validation (repeating the experiments for 10 different data partitions)

To make sure the results of experiments in 3.3.2 are valid, a cross-validation series of experiments was completed on the working dataset. In this series, the set of experiments described in 3.3.2 is repeated 10 times. Each time instead of considering the working dataset as a time series, the 2155 data points for the training dataset were picked through a random sampling. The remaining 1015 data points thus created the testing dataset, and were fed into the trained network also in random order, to check the ability of the model to react to unpredictable data patterns. The performance indices were averaged throughout the series of 10 experiments, and the results are summarized as mean RMSE and standard deviation across 10 experiments in the Figure 3.14 for the training data and in the Figure 3.15 for the testing data.

We can see that the results have very little difference across 10 experiments since standard deviations are around 1% of the respective mean values. It is visible from Figure 3.14 and Figure 3.15 that the performance index does not depend much on the changes in number of neurons in the hidden layer and learning rate. Also, it can be observed that, while on the original dataset the performance index on the training dataset was significantly better than on the testing dataset (Table 3.1), in cross-validation experiments, after obtaining averaged results, there is almost no difference between training and testing performance. From these results, the configuration performing best across 10 experiments can be defined. The best average results were achieved by the network with 11 neurons in the hidden layer trained with learning rate 0.03 (RMSE = 0.006041 on the testing dataset, pointed with an arrow on Figure 3.15). This is different from the best configuration found for the original dataset from Table 3.1 (5 neurons in the hidden layer, learning rate 0.015). However, it is difficult to assert that this network is the best for this dataset, since the data were fed in the model in a random order rather than in a form of a time series as in reality. The experiments described in the following sections were performed on the original dataset divided into the training and testing parts as a time series, without shuffling.

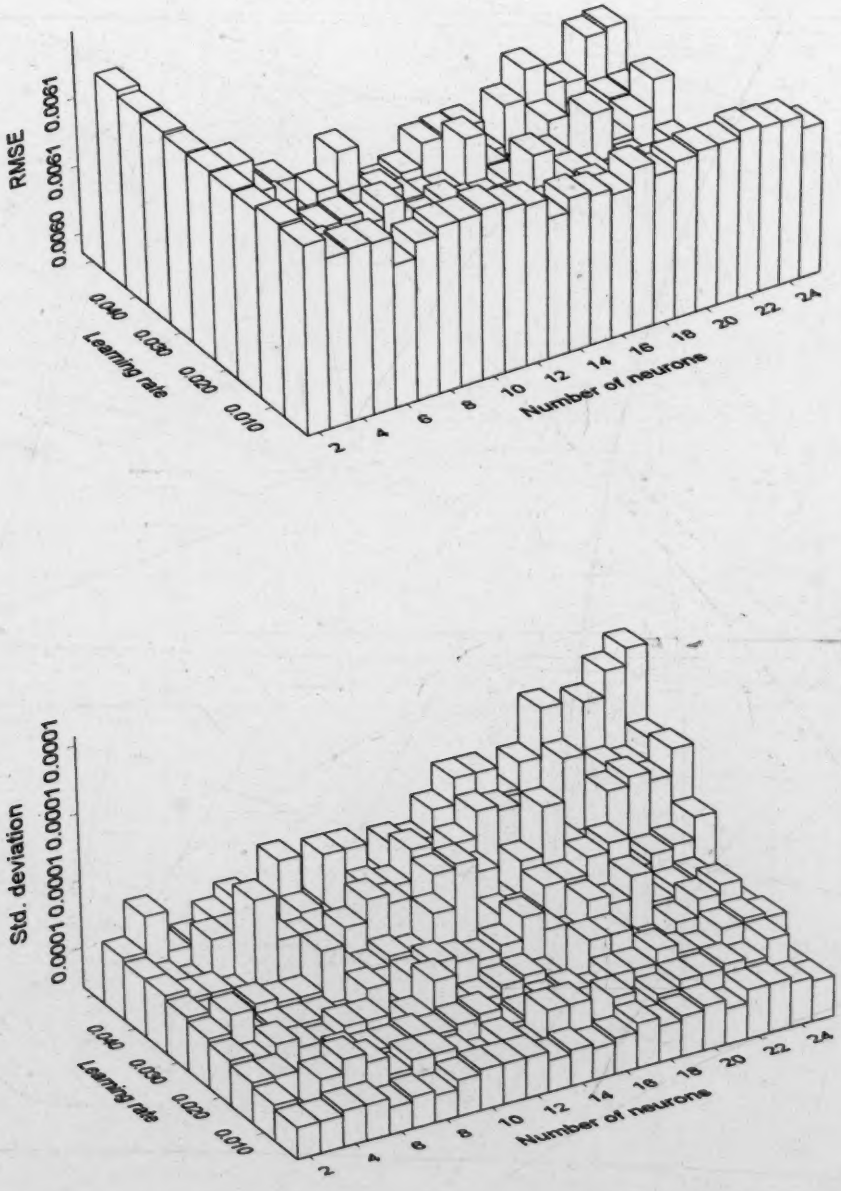


Figure 3.14 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for cross-validation experiments.

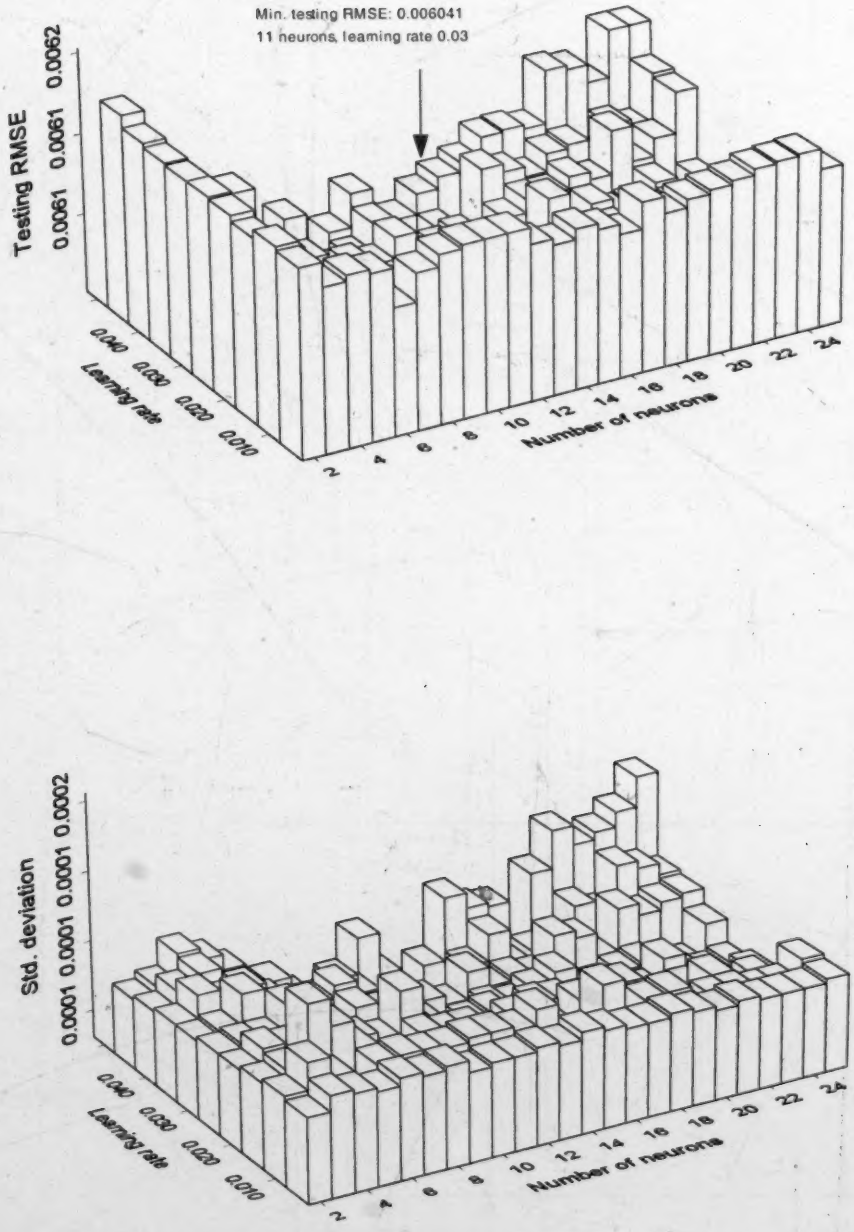


Figure 3.15 Testing RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for cross-validation experiments.

3.3.4. Comparing neural networks with different transfer functions

The experiment with linear output was set up the same way as in 3.3.2, only the transfer function of the network output layer was changed from sigmoid to linear. On the testing dataset, the best performance (RMSE = 0.005704) was obtained by the network with 3 neurons in the hidden layer, trained with the learning rate of 0.005. The network with sigmoid output layer performed slightly better than this model, the results for which are not presented in a table because there was no improvement. The network with the linear output layer also demonstrated the over-fitting effect: in the same manner as with sigmoid output layer, training RMSE declined with increasing the size of the hidden layer, but testing RMSE did not.

Then different transfer functions were tried for the hidden layer in the following experiment. The transfer function of the neurons of hidden layer was first changed from sigmoid (3.5) to arctangent (3.7) and then to hyperbolic tangent (3.8). The transfer function of the output layer was kept linear. The experiments in both cases were set up with the same parameters as in 3.3.2. In each set of experiments, the tables of experimental results similar to Table 3.1 were computed, to define the best performing configuration.

With an arctangent function (3.7) hidden layer, the best performance on the testing dataset (RMSE = 0.005796) was obtained by the network with 7 neurons in the hidden layer, trained with the learning rate of 0.005. This was slightly worse than for the networks with sigmoid hidden layer.

With a hyperbolic tangent function (3.8) hidden layer, the best performance on the testing dataset was RMSE = 0.006110, which was obtained by the network with 16 neurons in the hidden layer, trained with the learning rate of 0.015. This was, again, worse than for the networks with a sigmoid hidden layer, and also worse than for the networks with arctangent function hidden layer.

One should be careful about making generalizations on performance of the networks with different transfer functions. From this series of experiments, we can say only that on this particular dataset, these particular configurations performed slightly better with sigmoid transfer function than with arctangent, and both of them performed better than with hyperbolic tangent. These experiments just confirmed once again the popularity of sigmoid as a function of choice for the most neural network models of data analysis.

3.3.5. Introducing the momentum into the learning process

The experiments were repeated with introduced momentum term, for 8 different values of momentum (from 0.1 to 0.9, with the increment of 0.1). The set of experiments described in 3.3.2 was performed for each of these 8 momentum values. In that whole series of experiments, the best result on the testing dataset (RMSE = 0.005658) was achieved by the network with 5 neurons in the hidden layer, trained with the learning rate of 0.02 and momentum of 0.5. Experimenting with momentum helped to improve marginally the best result achieved by the neural network trained without momentum, however, this improvement can be considered negligible.

3.3.6. Analyzing the effect from introducing a second hidden layer

This model implements the two hidden layer network architecture presented on the Figure 3.6. In this series of experiments, all neurons in both hidden layers and output layer employed the sigmoid transfer function. Momentum term was not used, 10 values of the learning rate were tried (from 0.005 to 0.05, with the increment of 0.005), and the number of neurons in each of the two hidden layers was varied from 2 to 5. The results showed that adding a second hidden layer into the network brought some marginal improvement in performance, comparing to one hidden layer models (Table 3.2 vs. Table 3.1). The best result on the testing dataset (RMSE = 0.005677) was

obtained by the network with 3 neurons in the 1st hidden layer and with 5 neurons in the 2nd hidden layer, trained with the learning rate of 0.02 (marked with bold face in the Table 3.2).

Table 3.2 Performance index RMSE for each experiment with two hidden layers.

For training data:

1st	2nd	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
2	2	0.005603	0.005469	0.005488	0.005488	0.005495	0.005492	0.005491	0.005505	0.005512	0.005550
2	3	0.005437	0.005463	0.005478	0.005486	0.005509	0.005490	0.005471	0.00548	0.005503	0.005515
2	4	0.005441	0.005473	0.005492	0.005489	0.005502	0.005485	0.005496	0.005491	0.005498	0.005530
2	5	0.005442	0.005468	0.005490	0.005499	0.005487	0.005480	0.005484	0.005496	0.005502	0.005543
3	2	0.005588	0.005439	0.005445	0.005459	0.005453	0.005456	0.005499	0.005459	0.005445	0.005546
3	3	0.005433	0.005424	0.005429	0.005447	0.005448	0.005465	0.005443	0.005505	0.005459	0.005479
3	4	0.005441	0.005430	0.005432	0.005458	0.005444	0.005449	0.005424	0.005494	0.005472	0.005518
3	5	0.005429	0.005475	0.005440	**0.005462**	0.005450	0.005461	0.005454	0.005482	0.005514	0.005476
4	2	0.005618	0.005418	0.005436	0.005449	0.005439	0.005444	0.005458	0.005460	0.005472	0.005454
4	3	0.005423	0.005415	0.005430	0.005442	0.005448	0.005439	0.005432	0.005450	0.005454	0.005430
4	4	0.005407	0.005420	0.005425	0.005440	0.005455	0.005447	0.005413	0.005400	0.005456	0.005494
4	5	0.005423	0.005418	0.005433	0.005445	0.005437	0.005465	0.005431	0.005414	0.005476	0.005470
5	2	0.005539	0.005435	0.005438	0.005452	0.005457	0.005460	0.005436	0.005441	0.005444	0.005446
5	3	0.005426	0.005409	0.005424	0.005428	0.005416	0.005430	0.005426	0.005423	0.005437	0.005423
5	4	0.005496	0.005410	0.005427	0.005429	0.005416	0.005403	0.005433	0.005435	0.005414	0.005427
5	5	0.005425	0.005432	0.005429	0.005433	0.005459	0.005442	0.005388	0.005421	0.005383	0.005470

For testing data:

1st	2nd	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
2	2	0.006026	0.005782	0.005751	0.005741	0.005773	0.005765	0.005757	0.005778	0.005800	0.005873
2	3	0.005811	0.005775	0.005761	0.005761	0.005762	0.005767	0.005787	0.005789	0.005797	0.005816
2	4	0.005798	0.005762	0.005763	0.005745	0.005771	0.005769	0.005773	0.005796	0.005814	0.005816
2	5	0.005795	0.005765	0.005762	0.005764	0.005757	0.005770	0.005789	0.005805	0.005823	0.005843
3	2	0.005997	0.005729	0.005718	0.005694	0.005697	0.005721	0.005703	0.005715	0.005766	0.005850
3	3	0.005808	0.005699	0.005712	0.005702	0.005688	0.005700	0.005699	0.005750	0.005769	0.005720
3	4	0.005801	0.005727	0.005716	0.005689	0.005694	0.005720	0.005767	0.005693	0.005728	0.005768
3	5	0.005789	0.005768	0.005684	**0.005677**	0.005724	0.005705	0.005698	0.005730	0.005773	0.005699
4	2	0.006039	0.005747	0.005749	0.005703	0.005727	0.005699	0.005724	0.005727	0.005748	0.005758
4	3	0.005789	0.005720	0.005711	0.005691	0.005692	0.005733	0.005706	0.005706	0.005679	0.005732
4	4	0.005787	0.005709	0.005706	0.005713	0.005708	0.005698	0.005771	0.005805	0.005703	0.005767
4	5	0.005776	0.005707	0.005688	0.005704	0.005693	0.005719	0.005693	0.005788	0.005749	0.005734
5	2	0.005922	0.005728	0.005705	0.005715	0.005695	0.005708	0.005746	0.005787	0.005763	0.005765
5	3	0.005794	0.005727	0.005705	0.005699	0.005701	0.005713	0.005732	0.005716	0.005734	0.005853
5	4	0.005844	0.005722	0.005728	0.005695	0.005749	0.005747	0.005699	0.005729	0.005736	0.005716
5	5	0.005807	0.005700	0.005681	0.005740	0.005691	0.005738	0.005860	0.005756	0.005825	0.005786

First column denotes the number of neurons in the 1st hidden layer, second column – in the 2nd hidden layer, first row – learning rate.

3.3.7. Introducing feedback from the target outputs

In this experiment, a one hidden layer neural network with six additional inputs (on top of 8 regular data inputs) was implemented as on the Figure 3.7. The feedback from a previous cycle (target outputs corresponding to a previous data point) was introduced as described in sub-section

3.2.3.2. Each time the network was presented with a training data point, the 6 target outputs corresponding to that data point were submitted to these 6 extra inputs to be considered together with the next data point as if that next data point had 14 inputs instead of 8.

Adding the feedback brought significant improvement in performance comparing to the described above models without feedback (Table 3.3 vs. Table 3.1).

Table 3.3 Performance index RMSE for each experiment with one-cycle feedback.

For training data:

	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04
2	0.005043	0.00509	0.005109	0.005117	0.005122	0.005125	0.005124	0.005125
3	0.003797	0.003801	0.003801	0.005119	0.003826	0.003838	0.003837	0.003892
4	0.003800	0.003785	0.003808	0.003808	0.003818	0.003830	0.003835	0.003841
5	0.003795	0.003805	0.003811	0.003814	0.003807	0.003815	0.003846	0.003882
6	0.003780	0.003812	0.003796	0.003803	0.003827	0.003847	0.003872	0.003856
7	0.003798	0.003806	0.003788	0.003826	0.003812	0.003804	0.003878	0.003874
8	0.003801	0.003813	0.003802	0.003823	0.003818	0.003856	0.003886	0.003866
9	0.003803	0.003805	0.003801	0.003842	0.003833	0.003836	0.003811	0.003851
10	0.003799	0.003816	0.003808	0.003819	0.003794	0.003849	0.003862	0.003906
11	0.003799	0.003811	0.003835	**0.003802**	0.003824	0.003866	0.003851	0.003845

For testing data:

	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04
2	0.005375	0.005328	0.005314	0.005311	0.005316	0.005333	0.005341	0.005358
3	0.004054	0.004041	0.004012	0.005313	0.004027	0.004025	0.004073	0.004102
4	0.004065	0.004032	0.004032	0.004055	0.004021	0.004020	0.004073	0.004038
5	0.004046	0.004007	0.004037	0.004041	0.004011	0.004049	0.004048	0.004090
6	0.004030	0.004011	0.004007	0.004005	0.004000	0.004050	0.004031	0.004048
7	0.004018	0.003999	0.004026	0.004007	0.004038	0.004124	0.004076	0.004060
8	0.004011	0.004008	0.003999	0.004022	0.004024	0.004043	0.004058	0.004066
9	0.004007	0.004006	0.004020	0.004002	0.004009	0.004058	0.004034	0.004043
10	0.004061	0.004027	0.004024	0.004025	0.004028	0.004037	0.004068	0.004080
11	0.004044	0.004030	0.004001	**0.003982**	0.004020	0.004045	0.004018	0.004051

First column denotes the number of neurons in the hidden layer, first row – learning rate.

The best result on the testing dataset (RMSE = 0.003982) was achieved by the network with 11 neurons in the hidden layer, trained with the learning rate of 0.02 (marked with bold face in both parts of Table 3.3 visualized by Figure 3.16). The Figure 3.17, Figure 3.18, and Figure 3.19 plot the outputs y1 – y3 produced by the model vs. the target outputs.

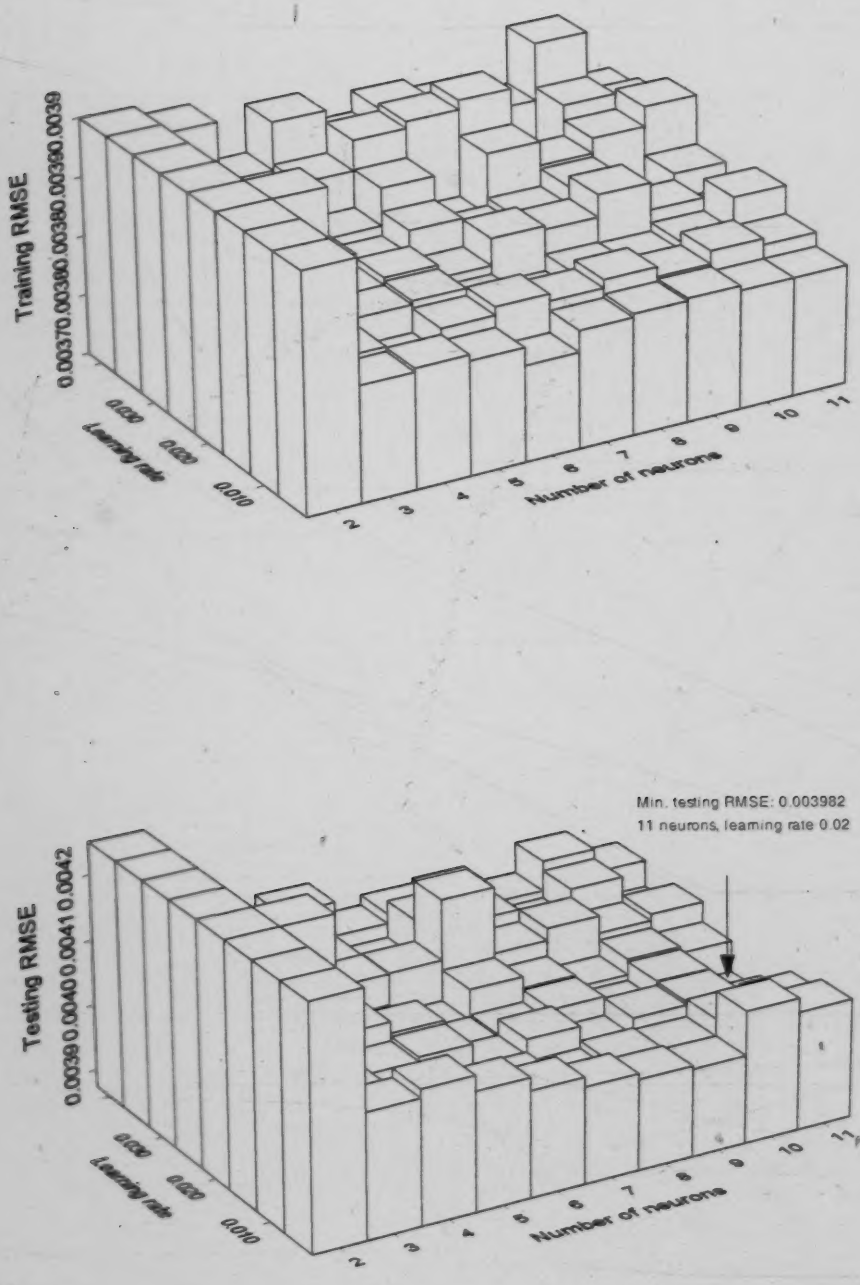


Figure 3.16 RMSE for feedback experiments as a function of number of hidden neurons and learning rate. Upper plot for training dataset, lower plot – for testing dataset.

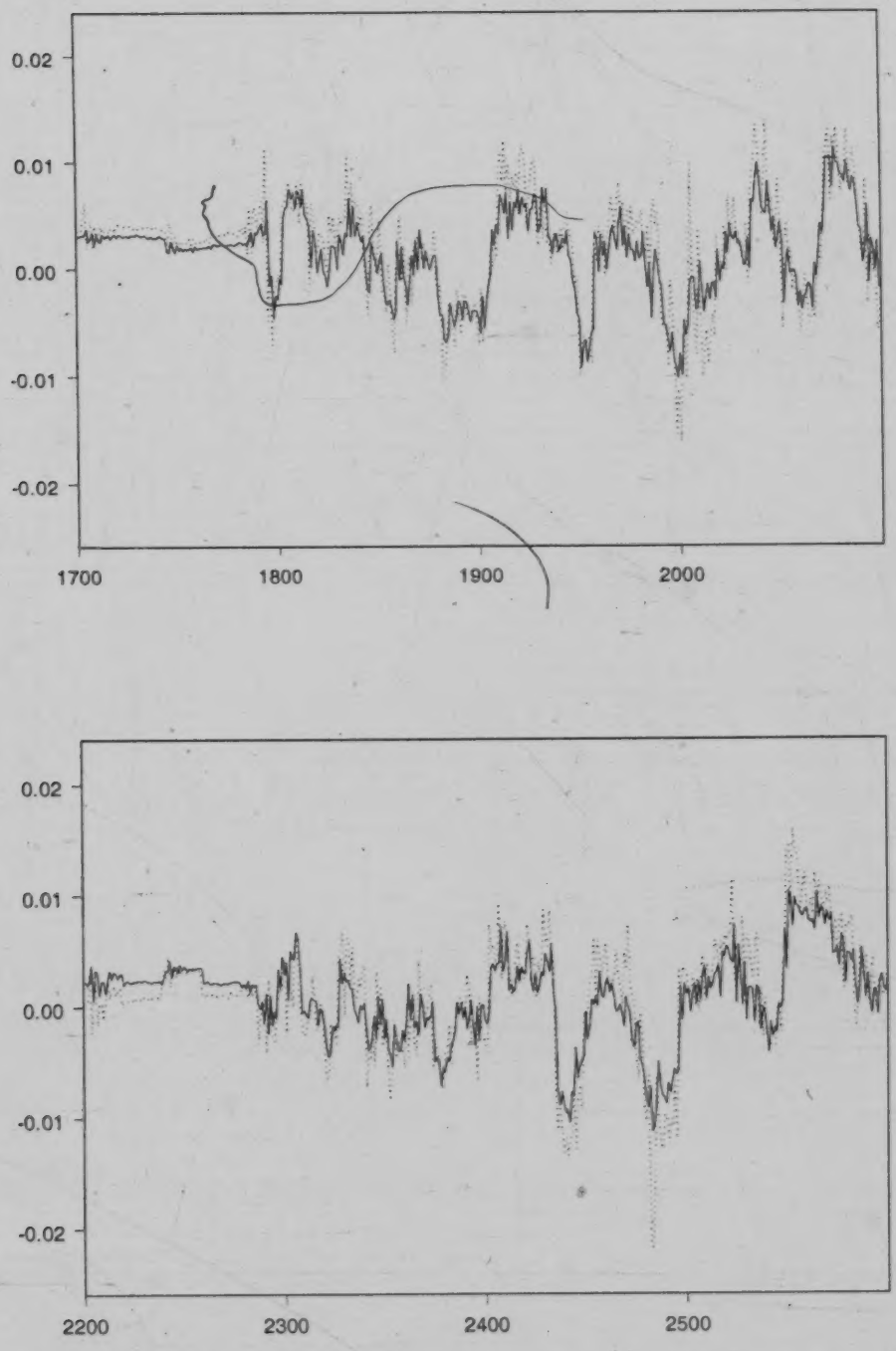


Figure 3.17 Floor X acceleration: dotted line – target, solid line – model output. Upper plot represents fragment of training data, lower plot – fragment of testing data. One-cycle feedback.

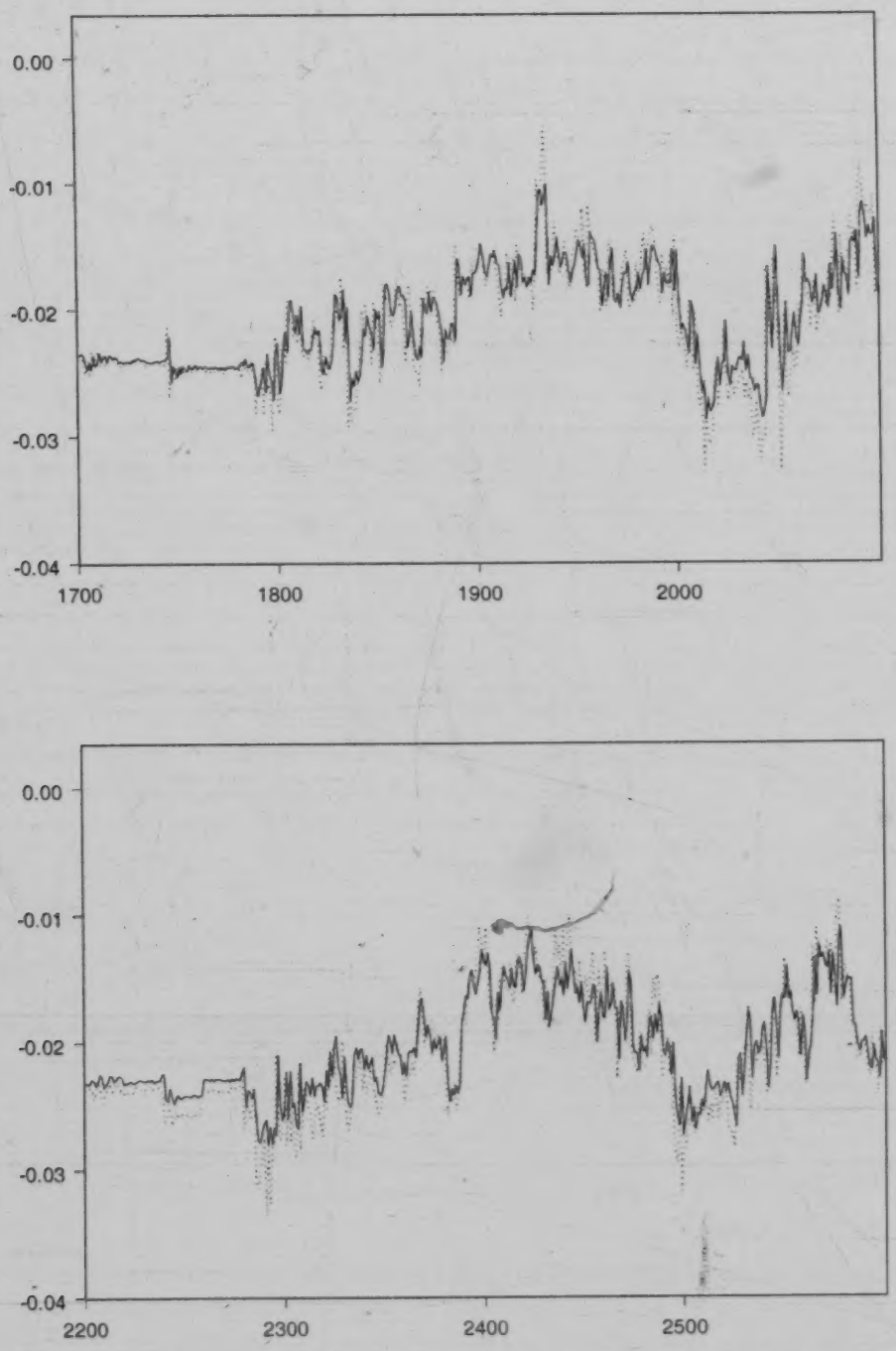


Figure 3.18 Floor Y acceleration: dotted line – target, solid line – model output. Upper plot represents fragment of training data, lower plot – fragment of testing data. One-cycle feedback.

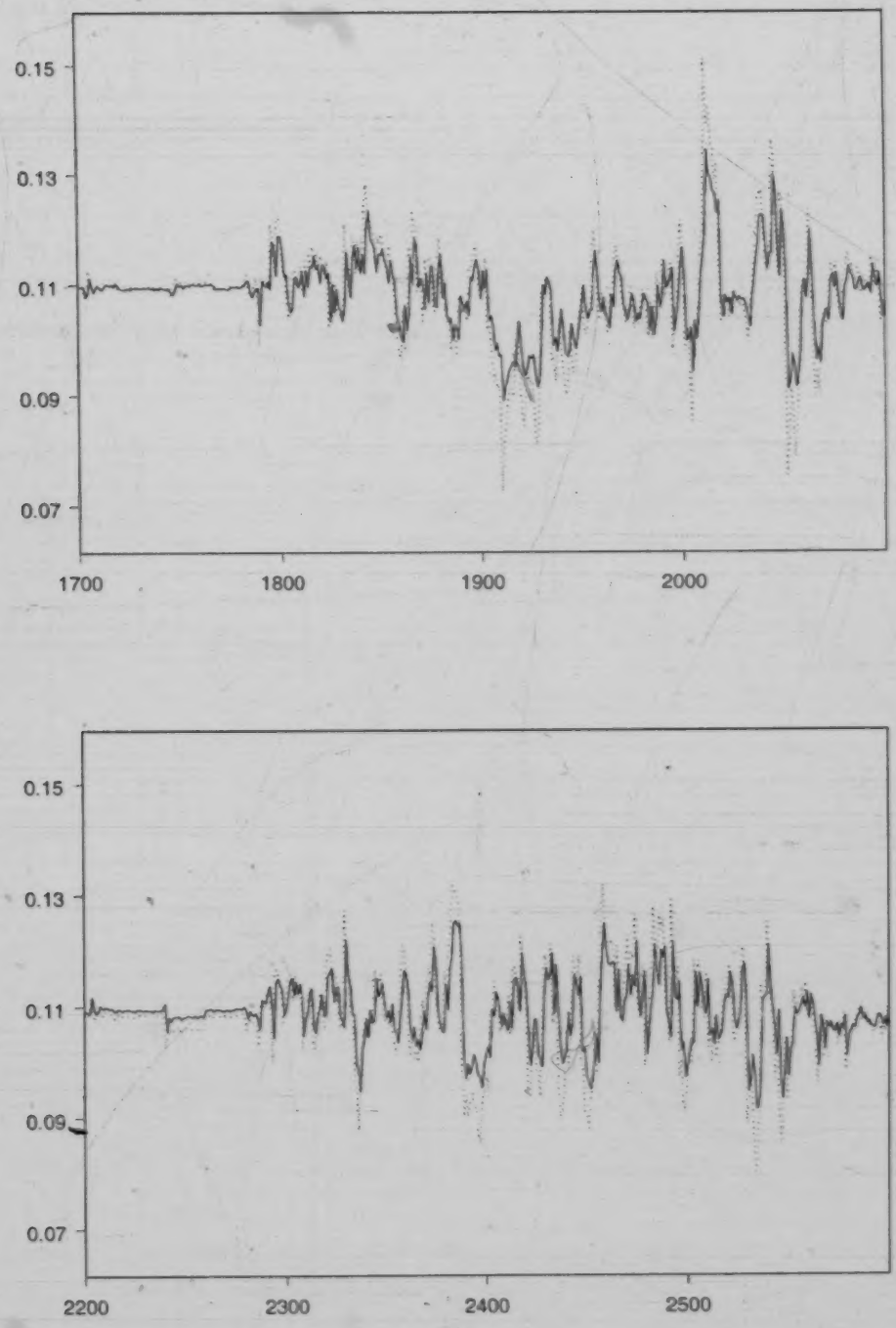


Figure 3.19 Floor Z acceleration: dotted line – target, solid line – model output. Upper plot represents fragment of training data, lower plot – fragment of testing data. One-cycle feedback.

The plots prove that the model with feedback performs well on all the six outputs, and most importantly, on the outputs y3 and y6 (vertical acceleration components) that usually caused problems for most of the models discussed above. For this model, correlation between target and predicted output values was 0.74 for the outputs y3 and y6, and exceeded 0.80 for the other four outputs. The explanation could be that the neural network considered not only actual inputs, but also the values of all six outputs from each previous data point as extra inputs. This provided the model with additional knowledge and evidently helped it to adjust weights in a way that lead to better approximating of all six outputs.

The cross-validation experiments for the model with feedback were set up in the following way. The extended dataset, containing extra inputs obtained from feedback, was partitioned into training and testing sections by 10 different ways. With temporal feedback, the sequence of data point is of crucial importance, because extra inputs of each next data point are taken from a current one. For this reason, the data were randomly shuffled not by single data points, but by chains of 10 sequential data points. This allowed us to create 10 different random partitions for training and testing parts, while preserving the order of data points within small sequences of data. However, since the whole time series is not preserved, the cross-validation results cannot demonstrate the best results achieved by the model with feedback on the original dataset.

Figure 3.20 and Figure 3.21 visualize the results of cross-validation experiments for neural network models with one-cycle feedback. Averaged results show the trend of declining RMSE with increase of number of neurons and decrease of learning rate. At the same time, the standard deviation increases with mean RMSE decrease. The best average result on the testing dataset (testing RMSE = 0.005199) was showed by a neural network with 11 neurons in the hidden layer trained with the learning rate of 0.005 (Figure 3.21).

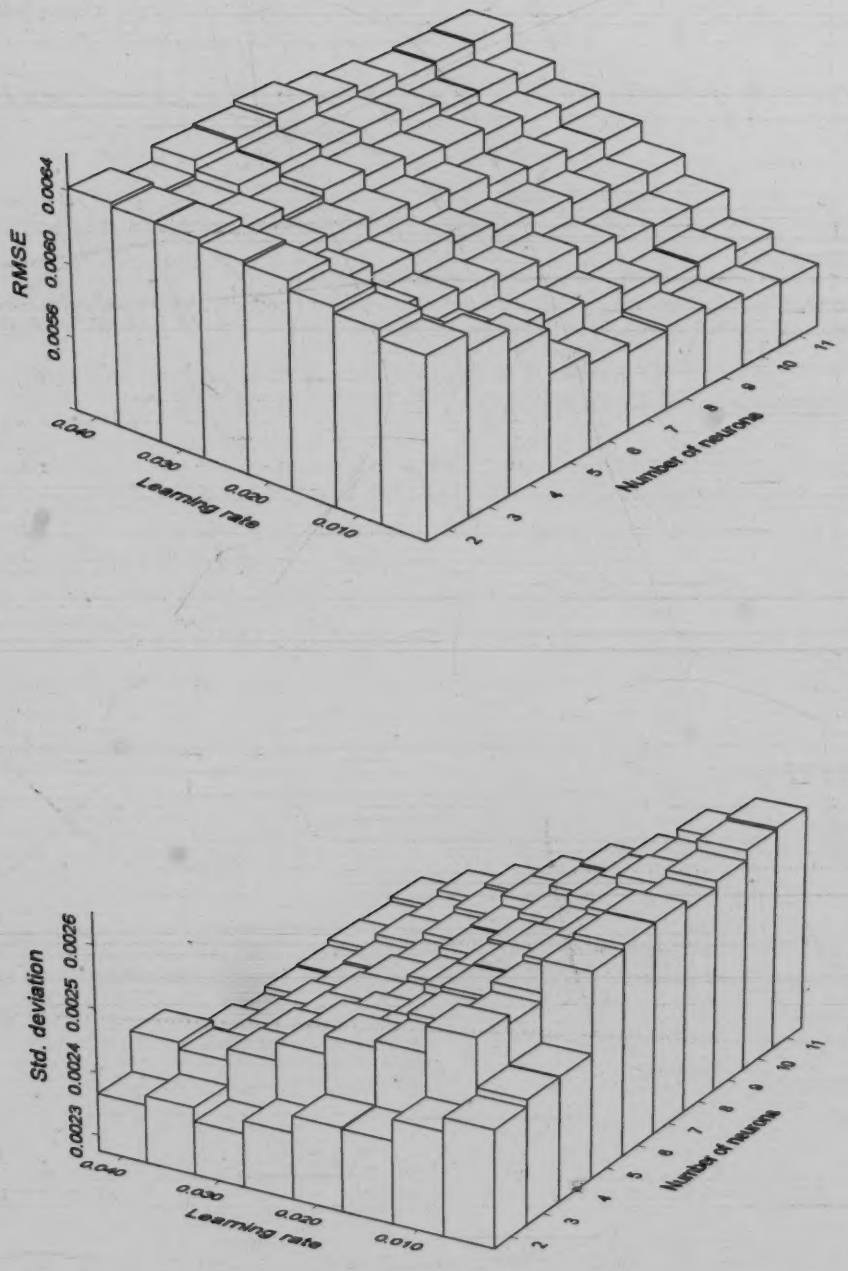


Figure 3.20 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for one-cycle feedback cross-validation experiments.

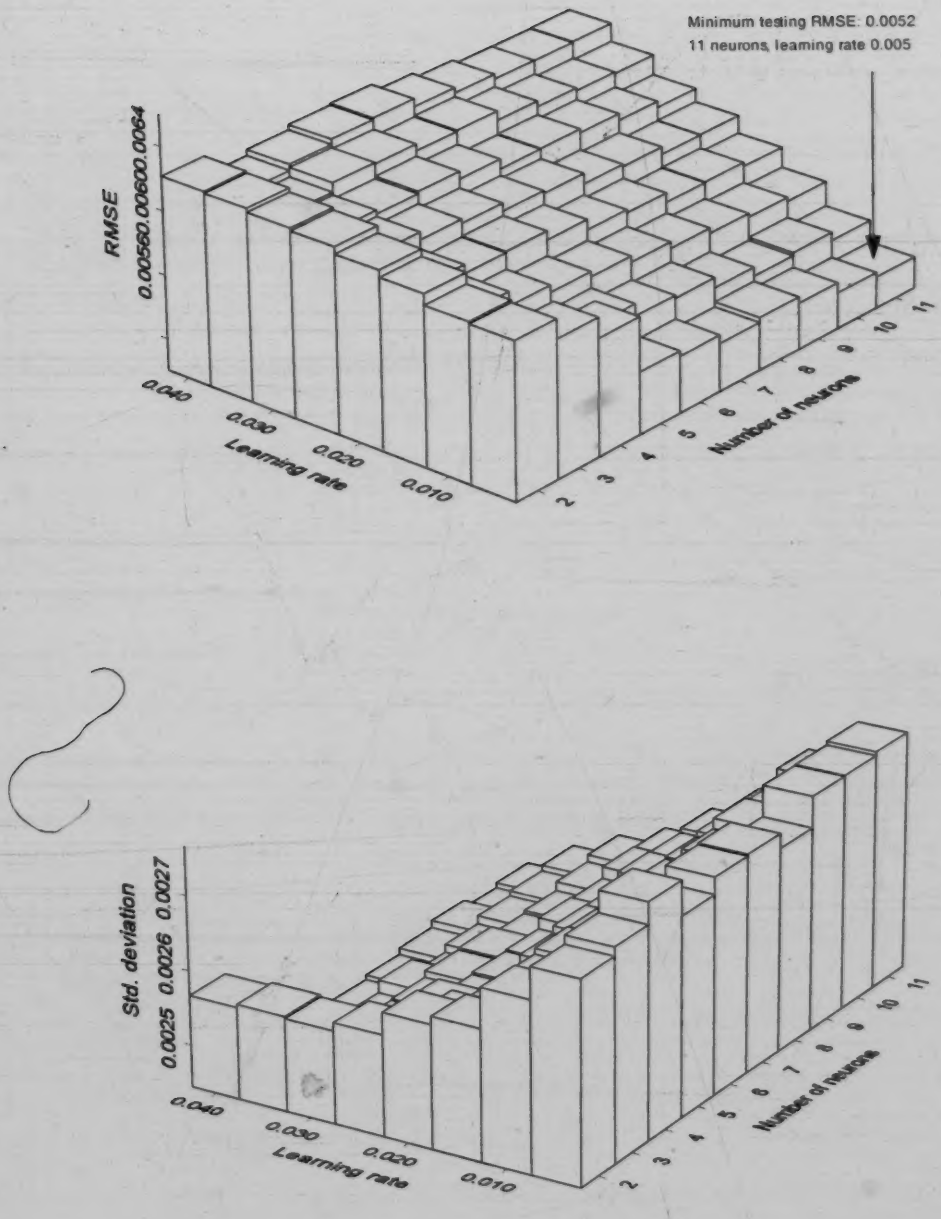


Figure 3.21 Testing RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for one-cycle feedback cross-validation experiments.

3.3.8. Experimenting with multi-cycle feedback

Besides experimenting with a feedback from target outputs taken from a previous cycle, the experiments with feedback were extended in a way to implement multi-cycle feedback. Here, the target outputs from two previous cycles were considered to be used as a feedback, i.e. adding 12 extra inputs to the model. Two-cycle feedback also produced very good results vs. the model without feedback (Table 3.4 vs. Table 3.1). Its performance was slightly better than for the model with one-cycle feedback (Table 3.4 vs. Table 3.3).

Table 3.4 Performance index RMSE for each experiment with two-cycle feedback.

For training data:

	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04
2	0.005419	0.005051	0.005061	0.005074	0.005083	0.005084	0.005086	0.005066
3	0.003775	0.003706	0.003734	0.003746	0.003747	0.003767	0.005079	0.003791
4	0.003722	0.005061	0.003738	0.003748	0.003759	0.003776	0.003798	0.003816
5	0.003737	0.003715	0.003729	0.003752	0.003767	0.003807	0.003794	0.003806
6	0.005048	0.003726	0.003737	0.003761	0.003754	0.003784	0.003793	0.003794
7	0.003729	**0.003731**	0.003738	0.003753	0.003770	0.003788	0.003782	0.003788
8	0.003718	0.003734	0.003755	0.003762	0.003776	0.003797	0.003827	0.003808
9	0.003731	0.003719	0.003750	0.003768	0.003785	0.003762	0.003804	0.003797
10	0.003727	0.003749	0.003732	0.003773	0.003753	0.003804	0.003796	0.003821
11	0.003722	0.003740	0.003763	0.003750	0.003778	0.003770	0.003789	0.003815

For testing data:

	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04
2	0.005804	0.005282	0.005257	0.005264	0.005276	0.005287	0.005307	0.005310
3	0.004077	0.00401	0.003999	0.003999	0.004006	0.004023	0.005322	0.003993
4	0.004044	0.005261	0.003982	0.004009	0.003984	0.004010	0.004032	0.004046
5	0.004027	0.004004	0.003994	0.003976	0.004010	0.004021	0.004017	0.004041
6	0.005282	0.003958	0.004006	0.004005	0.004006	0.004024	0.004024	0.003994
7	0.003991	**0.003950**	0.003988	0.003982	0.003980	0.004015	0.003991	0.004029
8	0.004004	0.003950	0.003986	0.003990	0.004018	0.003975	0.004057	0.004020
9	0.003986	0.003972	0.003988	0.003965	0.004020	0.003989	0.003996	0.004013
10	0.004008	0.003978	0.003976	0.003976	0.003984	0.004040	0.004011	0.004021
11	0.003976	0.003956	0.003950	0.003967	0.003973	0.003980	0.003996	0.004077

First column denotes the number of neurons in the hidden layer, first row – learning rate.

The network with 7 neurons in the hidden layer, trained with the learning rate of 0.01, reached the best performance on the testing dataset RMSE = 0.003950 (marked with bold face in both parts of

the Table 3.4). During its training, the network was able to overcome a plateau and a local minimum and finally converged after 4000 iterations to the level of training RMSE = 0.003730 (Figure 3.22).

Overall, this model delivered the best result among all the models tried in the experiments with the back-propagation neural networks.

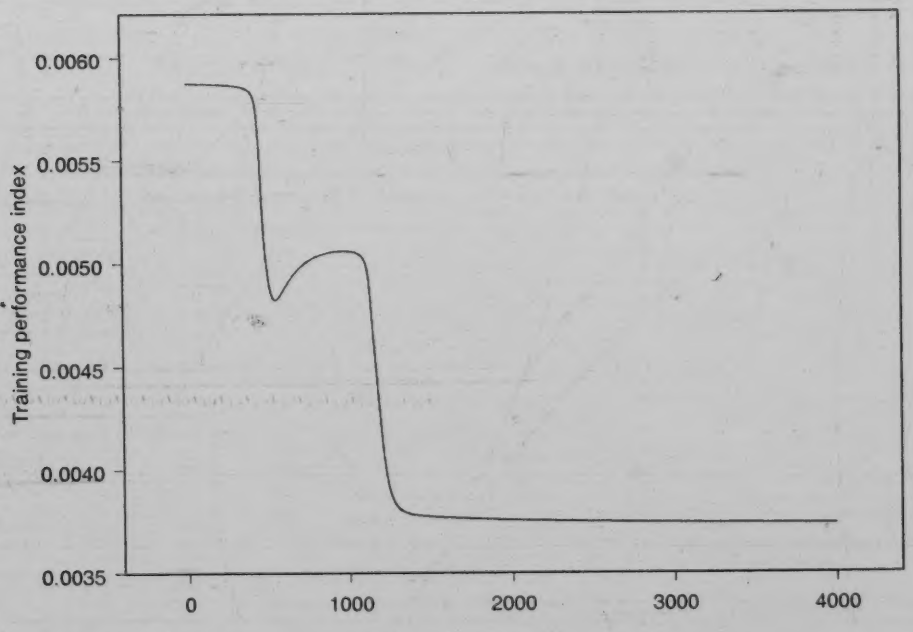


Figure 3.22 Convergence of the training performance index for the model with two-cycle feedback.

The cross-validation results for the model with two-cycle feedback are presented in Figure 3.23 and Figure 3.24. Same as for one-cycle feedback, mean RMSE decreases and the standard deviation increases with increase of number of neurons and decrease of learning rate. The best average result on the testing dataset (testing RMSE = 0.005245) was showed by a neural network with 10 neurons in the hidden layer trained with the learning rate of 0.005.

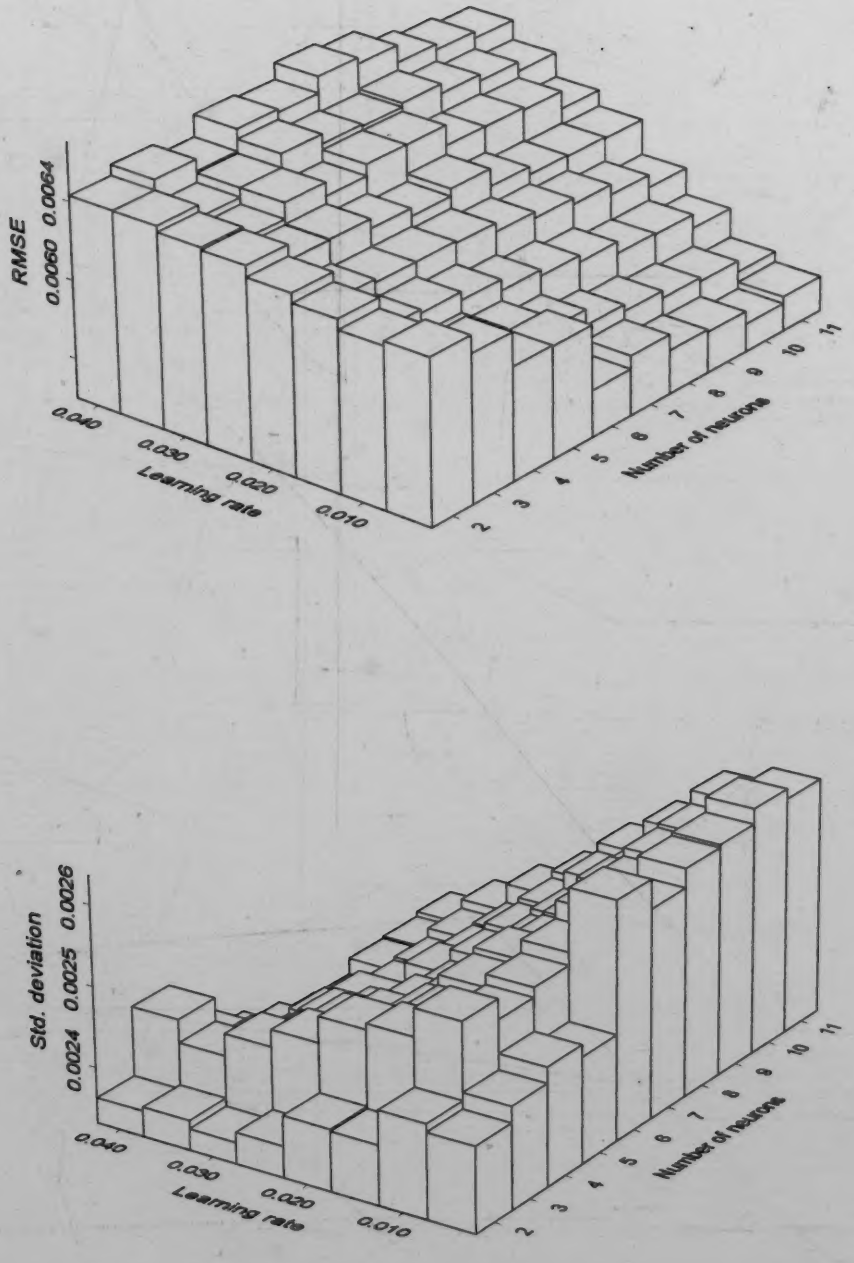


Figure 3.23 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for two-cycle feedback cross-validation experiments.

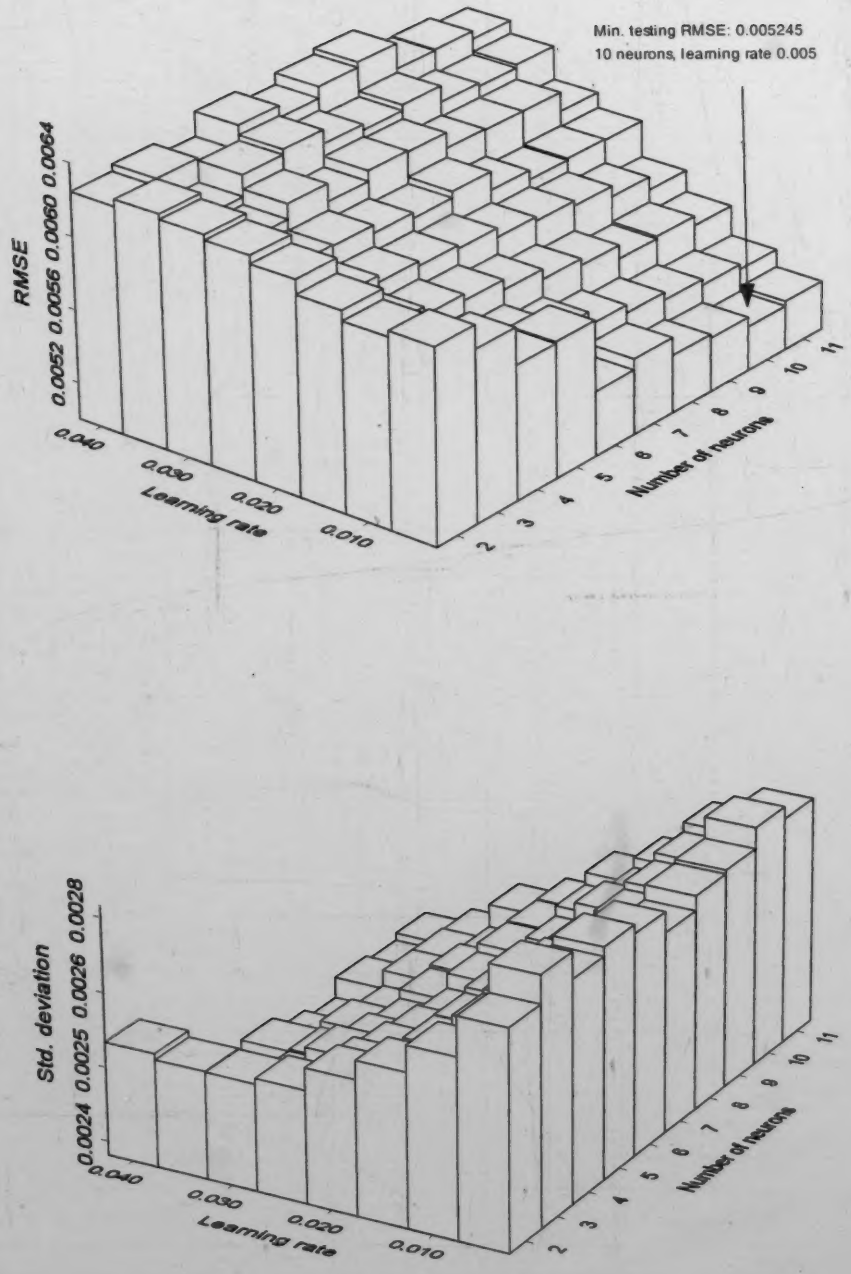


Figure 3.24 Training RMSE (upper plot) and standard deviation (lower plot) as a function of number of hidden neurons and learning rate for two-cycle feedback cross-validation experiments.

The two-cycle feedback gained practically no improvement against one-cycle feedback. As for the three-cycle feedback, implemented as 18 extra inputs taken from target outputs from three previous cycles, it not only did not improve the results, but caused deterioration of the model performance. Furthermore, it complicated the model and increased the execution time. The model learnt well the outputs y2 and y5, but performed worse on the outputs y1 and y4, and again completely failed on the outputs y3 and y6.

3.4. Experimenting with feedback separately on different outputs

To understand the difference between the model performance on the outputs y1, y2, y4, and y5 from one side (acceleration dimensions X and Y), and on the outputs y3 and y6 from the other side (acceleration dimension Z), these outputs were separated from each other. Instead of analyzing all the six outputs of the working dataset together, the experiments first were conducted on the dataset containing all 8 inputs and only the outputs y1, y2, y4, and y5. The performance results (for 5 neurons in the hidden layer and learning rate of 0.0015) were as follows:

Model type	No feedback	1-cycle feedback	2-cycle feedback	3-cycle feedback
Training RMSE	0.003522	0.002736	0.002647	0.002631
Testing RMSE	0.003792	0.002762	0.002656	0.002639

Then the same experiments were repeated on the dataset containing same inputs and only the outputs y3 and y6, with the following results:

Model type	No feedback	1-cycle feedback	2-cycle feedback	3-cycle feedback
Training RMSE	0.007966	0.005331	0.005317	0.005297
Testing RMSE	0.008327	0.005747	0.005761	0.005763

We can see that in all these experiments, the results on the outputs y3 and y6 are always worse than for the outputs y1, y2, y4, and y5 (RMSE about twice as higher). This observation was true for all the other models as well. RMSE around 0.0080 for the outputs y3 and y6 meant that a model failed to learn from these outputs (see Figure 3.12). The decline of the RMSE to the level of around 0.0050 was evidence of significant improvement in the model performance on these outputs and quite reasonable results, both on the training and testing data (see Figure 3.19). This

improvement was achieved by introducing feedback, and we can see that two- and three-cycle feedback added almost no improvement versus one-cycle feedback. It was the feedback idea per se that made the biggest difference versus all the models without feedback

3.5. Conclusions

It made sense to start experimenting with the simplest possible configuration, according to the Occam's razor principle. The results of experiments with the simple architecture containing one hidden layer and neurons implementing most common sigmoid transfer function in both hidden and output layers, were acceptable for 4 out of 6 outputs. The number of neurons in the hidden layer, as well as the learning rate modifications did not make a significant influence on the model performance.

Changing the sigmoid transfer function in the output layer for linear did not bring any improvement, the performance became slightly worse. The same effect was observed when changing the sigmoid transfer function in the hidden layer to some other types of functions. With the arctangent transfer function, the network performance became slightly worse than with the sigmoid transfer function. The hyperbolic tangent transfer function produced the results that were slightly worse than with the arctangent transfer function.

Experimenting with introducing a momentum term into training brought marginal improvement compared to the best result achieved by the neural network trained without momentum, however, this improvement can be considered negligible. Also, adding a second hidden layer into the network brought similar marginal improvement in performance when compared to one hidden layer models.

In contrast with negligible difference in performance in all those experiments, adding feedback from target outputs taken from a previous cycle brought significant improvement in performance

when compared to the models without feedback. This model performed well on all six outputs, while the other models showed acceptable results only on 4 outputs out of 6 and failed on the two most important ones. Adding the feedback from one more previous cycle brought marginal improvement comparing to one-cycle feedback. Experimenting further in this direction revealed the evidence that 3-cycle feedback did not bring further improvement to the model performance, while complicating and slowing down the experiments.

The experiments proved that the model with a feedback achieved the best performance among all the other models. It can be concluded that 1-cycle feedback was enough to deliver practically the same results as with 2- and 3-cycle feedback models, without excessive complication of the model and increasing of the execution time.

The connections between layers for the best performing neural network model (11 neurons in one hidden layer, one-cycle feedback) are presented in Table 3.5 where the first line denotes the neurons of the hidden layer, the first column – the inputs and outputs connected to those neurons.

Table 3.5 Weights of connections between layers for the best performing neural network model.

Neurons	1	2	3	4	5	6	7	8	9	10	11
Input 1	-0.075	0.365	0.913	0.675	0.389	0.537	-0.342	0.154	0.086	0.052	0.559
Input 2	-0.218	0.751	0.470	0.752	1.327	0.788	0.991	0.368	1.206	0.304	1.046
Input 3	-0.386	0.459	0.612	1.319	0.626	0.658	0.457	-0.195	0.668	0.015	0.762
Input 4	0.097	0.363	0.664	0.313	0.372	0.169	-0.496	0.132	0.869	0.010	0.325
Input 5	-0.114	0.365	0.108	0.413	0.673	0.543	-0.810	0.731	0.811	0.659	0.831
Input 6	-0.793	1.034	0.499	0.672	-0.066	0.427	0.270	-0.093	0.728	0.598	0.595
Input 7	0.836	0.032	0.061	0.861	0.527	0.141	0.193	0.049	0.888	-0.929	0.280
Input 8	0.028	0.642	0.799	0.869	0.103	0.742	-0.250	0.039	0.549	-0.146	0.678
Bias input	-1.677	1.008	0.958	1.607	1.042	1.156	1.101	0.080	1.257	0.061	0.952
Feedback 1	-0.249	0.665	-0.047	0.560	1.171	0.456	0.822	-0.445	0.387	-0.616	0.790
Feedback 2	0.759	0.716	-0.058	0.538	0.596	0.695	0.250	2.463	0.797	-0.701	0.878
Feedback 3	2.326	0.911	0.124	0.242	0.980	0.065	0.570	-1.060	0.841	1.445	0.935
Feedback 4	-1.585	0.317	0.680	0.479	0.332	0.884	0.228	0.645	0.300	1.285	0.054
Feedback 5	0.003	0.912	0.074	0.334	0.912	0.571	0.369	-1.495	0.229	-0.562	0.782
Feedback 6	0.757	0.916	0.420	1.072	0.717	0.689	0.594	-0.092	0.914	-0.391	0.915
Output 1	1.104	0.444	0.468	1.070	1.457	0.498	1.779	-0.699	0.880	-3.415	0.511
Output 2	1.084	-0.110	0.253	0.794	0.282	0.534	0.084	3.192	0.568	-1.107	-0.111
Output 3	3.158	0.349	0.380	1.071	0.604	0.462	1.078	-0.695	1.025	2.339	0.952
Output 4	-1.837	1.489	1.281	1.232	1.109	1.107	1.602	0.474	1.225	3.651	0.969
Output 5	-1.426	1.721	1.748	1.223	1.482	1.499	1.319	-2.663	1.029	-0.050	1.027
Output 6	3.177	-0.222	0.811	0.065	-0.315	0.177	0.796	-0.845	-0.134	2.252	-0.088

Chapter 4. Local models based on data clustering

This chapter presents the concept of data partition into clusters. Clustering is based on looking for similarities between different data points. The notion of distance is employed as a criterion of similarity. An algorithm of data fuzzy clustering (known as FCM algorithm) is described. Several types of models are discussed that are built on the concept of the receptive fields corresponding to the clusters created by the FCM algorithm. The following types of model were tried in the experiments: local linear regressions, Takagi – Sugeno model, and RBF networks. For each of these types, multiple experiments were conducted with different numbers of clusters. Also, some parameters of the models were changed in the course of the experiments, such as: FCM termination criterion, methods of data allocation into clusters, types of radial basis functions, training RBF network versus computing the weights through the least-square solution, etc. Cross-validation experiments for 10 different random splits of the dataset into training and testing parts were also conducted for local linear regressions and RBF networks. For each model and each experiment, the performance index was calculated for training and testing parts of the dataset. The results delivered by different models are compared to each other and to the results obtained with a linear regression built for the entire dataset.

4.1. Concept of clusters as receptive fields

Big datasets are often difficult to analyze because of their heterogeneous structure demonstrating quite different patterns of behavior in the different areas of the dataset. When a single model is built on the whole dataset, it inevitably tries to find something average between the different and often contradictory patterns existing in the different areas of the dataset. Such attempts to find an average solution for a diverse dataset may compromise the total performance of the model as a whole.

The idea of clustering looks for solutions taking into account this heterogeneous structure of the most datasets. It proposes to arrange a collection of data into a small number of groups (clusters) so that the elements that are similar become allocated to the same group, and the elements that are quite distant should be placed into separate categories [3]. In fact, clustering is a part of pattern recognition. The patterns are organized by taking into account certain optimization criterion. The arrangement of the patterns is directly implied by minimization of this criterion.

Clustering has been often exercised during a pre-processing phase in the design of data analysis models like RBF neural networks [6], or Takagi-Sugeno models [8]. The primary aim of the clustering algorithm is to set up an initial distribution of the receptive fields across the space of the input variables [7].

The dominant factor in the process of clustering is the distance between two patterns, i.e. the notion defining their similarity or resemblance. The role of the distance function is to quantify a notion of similarity: the shorter the distance between two patterns, the higher the level of their similarity [3]. One of the most popular distance functions, Euclidian distance, is used throughout this chapter:

$$\| \mathbf{x} - \mathbf{y} \| = \sqrt{\sum_{i=1}^{n} | x_i - y_i |^2} \tag{4.1}$$

where

- **x** and **y** are two different data points (patterns)
- $\| \mathbf{x} - \mathbf{y} \|$ denotes Euclidian distance between data points **x** and **y**
- n is dimensionality of the dataset
- x_i is one of the n dimensions of the data point **x**
- y_i is the corresponding dimension of the data point **y**
- $| x_i - y_i |$ denotes distance between data points **x** and **y** projected along dimension i

In general, clustering methods fall into two dominant classes [3]:

1. hierarchical clustering employing strait-forward aggregative and divisive methods;
2. objective function optimization methods where partitioning of the data patterns is performed through developing and adjusting a partition matrix in such a way that a certain performance index (optimization criterion) becomes minimized.

In the FCM algorithm belonging to the second class of methods, the basic objective function to be minimized assumes the form of a sum of distances ($\|\cdot\|$) between the data points (patterns) and the prototypes (centroids) of clusters [7]:

$$Q = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^{p} \| \mathbf{x}_k - \mathbf{v}_i \|^2 \tag{4.2}$$

where

- $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are the data in the form of n-dimensional vectors distributed in $\mathbf{R}^n$,
- $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c$ are the prototypes of clusters,
- c is the number of clusters
- N is the number of the data points in the dataset

- $[u_{ik}]$ forms a partition matrix of dimension c×N, p>1

The partition matrix describes a partition of the data into a predetermined number of clusters. Each entry of it represents a value describing membership of one particular data point in one particular cluster. The partition matrix has two basic properties [12]:

$$0 < \sum_{k=1}^{N} u_{ik} < N \qquad \text{for } i = 1, 2, \ldots, c \tag{4.3}$$

$$\sum_{i=1}^{c} u_{ik} = 1 \qquad \text{for } k = 1, 2, \ldots, N \tag{4.4}$$

The property (4.3) asserts that each cluster is non-empty, and the property (4.4) – that each element belongs to only one cluster (for set oriented clustering), or the sum of membership values to all clusters is equal to 1 (for fuzzy clustering) [3].

A popular example of set oriented clustering is k-means clustering algorithm [11]. Each data point has its own column in the partition matrix where its membership in a particular cluster is marked by 1 in the line corresponding to that cluster, while all the other entries contain 0. For each new analyzed data point, an additional column of partition matrix is computed, based on the distance of this data point to all already existing clusters, and the data point becomes allocated to the nearest (in terms of employed distance function) cluster.

With the data allocated into groups by similarity, each group can employ a separate model of analysis and prediction of the data. Then the model parameters can be specially adjusted for this specific group of data. This allows us to build more precise local models versus unified general model embracing the whole available dataset.

When such multi-component model is used for prediction, the new data is first classified from the point of view of where it might belong in terms of patterns defined for the historically collected data. The classification algorithm decides which group is the most appropriate for the new data to be allocated to, and then the local model, with the parameters adjusted specifically for that group, is applied to make predictions for this new data. In that sense, different parts of the model operate as receptive fields, since they automatically recognize the new data as similar or not similar to what they are responsible for, and then only one of these receptive fields receives the data for processing (Figure 4.1).

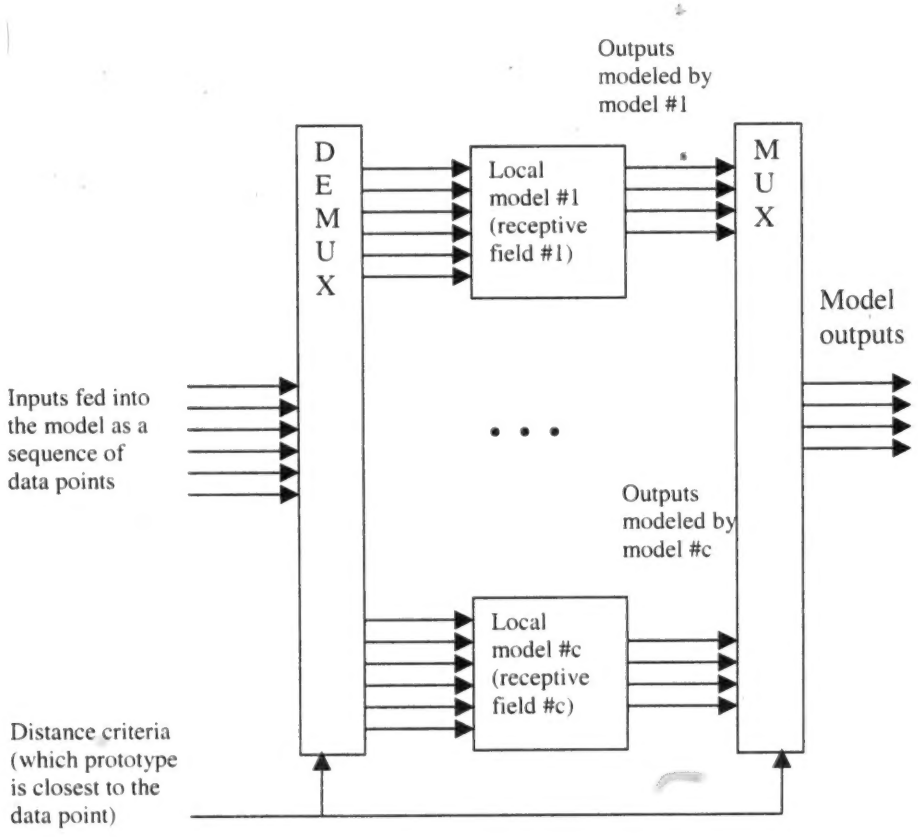


Figure 4.1 Local models as receptive fields to process the data

4.2. Fuzzy C-Means clustering algorithm

The rational of fuzzy clustering is that instead of assigning each data point to one specific cluster as in k-means algorithm [11], the data point is assigned to all existing clusters with different membership values that all sum up to 1, according to the second property of the partition matrix (4.4).

Once the number of clusters is defined, the FCM (Fuzzy C-Means) clustering algorithm [12] can be applied to obtain the cluster prototypes. Before the FCM algorithm can be applied, all values in each input of the original dataset must be normalized into the interval [0,1], so that none of the inputs dominates over the others in the process of computing the distances.

At first, the partition matrix is initialized with random values in the interval [0, 1], and then each column of the partition matrix must be scaled to satisfy the condition of its sum being equal to 1 (the property 4.4). This means that each data point belongs to all clusters with the total sum of all (c) membership values equal to 1.

Thus, the FCM clustering algorithm begins with a random partition matrix and then iteratively converges to particular values of cluster prototypes and the partition matrix, stopping when there is no more improvement when adjustments are made to the prototypes. This iterative process optimizes the objective function (4.2).

At each iteration, the FCM algorithm computes the matrix of Euclidian distances between each data point and each cluster prototype. The distance between data point $\mathbf{x}_k$ (k = 1, 2, ..., N) and prototype $\mathbf{v}_i$ (i = 1, 2, ..., c) is computed as a Euclidian sum of the distances along each of n dimensions of the input space:

$$\| \mathbf{x}_k - \mathbf{v}_i \| = \sqrt{\sum_{j=1}^{n} | x_{kj} - v_{ij} |^2} \qquad (4.5)$$

Then, based on these distances, the FCM algorithm updates the partition matrix [3]:

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\| \mathbf{x}_k - \mathbf{v}_i \|}{\| \mathbf{x}_k - \mathbf{v}_j \|} \right)^{\frac{2}{m-1}}} \tag{4.6}$$

The prototype vector $\mathbf{v}_i$ (i = 1, 2, ..., c) is updated along each dimension j (j = 1, 2, ..., n) at each iteration as the weighted average of individual data points [3]:

$$v_{ij} = \frac{\sum_{k=1}^{N} u_{ik}^{m} x_{kj}}{\sum_{k=1}^{N} u_{ik}^{m}} \tag{4.7}$$

The fuzzification factor m for cluster partition influences the form of membership functions. For lower values of the fuzzification factor, the resulting membership functions become localized around 0 or 1. In this case they resembe characteristic functions of sets with fewer elements having intermediate membership values. With increasing values of the fuzzification factor, the membership functions tend to show more local minima [3]. In this research, the fuzzification factor is set to m = 2.

FCM termination criterion means that the iterative process stops after the performance index computed in the current iteration differs from the one computed in the previous iteration less than by the value of the termination criterion. The performance index is computed as a value of the objective function (4.2), i.e. sum of all squared distances weighted by corresponding elements of the partition matrix.

The essence of the performance index is that bigger distances must be compensated by smaller values of the corresponding elements in the partition matrix, so with each next iteration, the performance index should decrease because of two reasons:

1. partition matrix is updated in the way that bigger distances produce smaller membership values of corresponding elements in partition matrix, and vice versa;
2. prototypes are updated as cluster centres of individual data points weighted by adjusted ("improved") values of partition matrix, thus producing smaller sum of all distances.

As a result of the FCM algorithm, each n-dimensional data point $\mathbf{x}_k$ (k = 1, 2, ..., N) is fuzzily assigned to all (c) clusters with different membership values. The cluster prototypes $\mathbf{v}_i$ (i = 1, 2, ..., c) are adjusted to reflect the centres of data clusters in terms of minimized objective function, and also the partition matrix is adjusted to represent the membership values with which each data point belongs to each cluster.

4.3. Data visualization as a pre-processing tool before building a model

Before defining the model parameters, it makes sense to undertake some attempts of data visualization. Although the working dataset is multidimensional (8 inputs and 6 outputs), pairs of its dimensions can be represented by the plots (Figure 4.2, Figure 4.3, Figure 4.4, Figure 4.5, Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9, Figure 4.10, Figure 4.11). Several other techniques can be applied to try to visually reveal some basic structure existing in the dataset, which might give some insight and help in defining the model basic parameters, such as number of clusters, their radii, etc. However, not necessarily it would be clear from visualization plots: we cannot say that five clusters shown here represent the number more appropriate for this dataset than any other number of clusters would be.

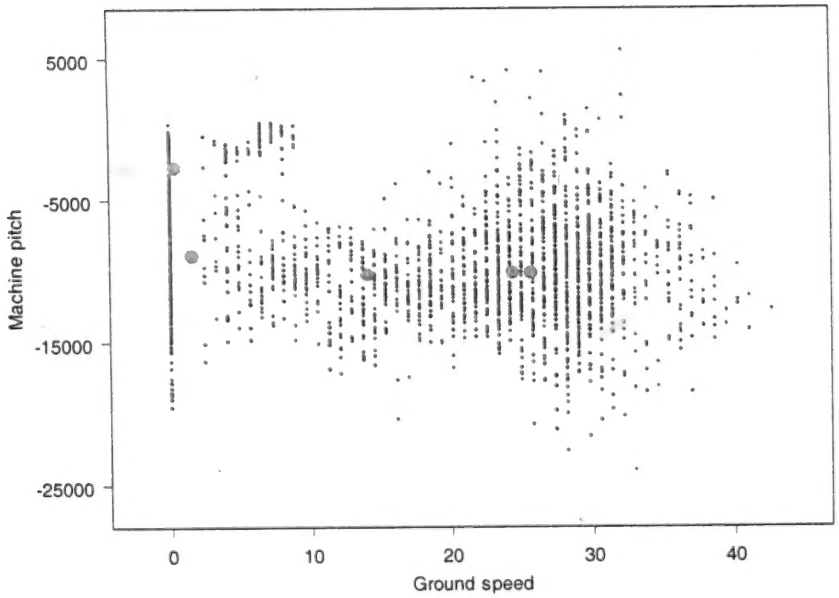


Figure 4.2 Scatter plot for the pair of inputs x2 (machine pitch) vs. x1 (ground speed)

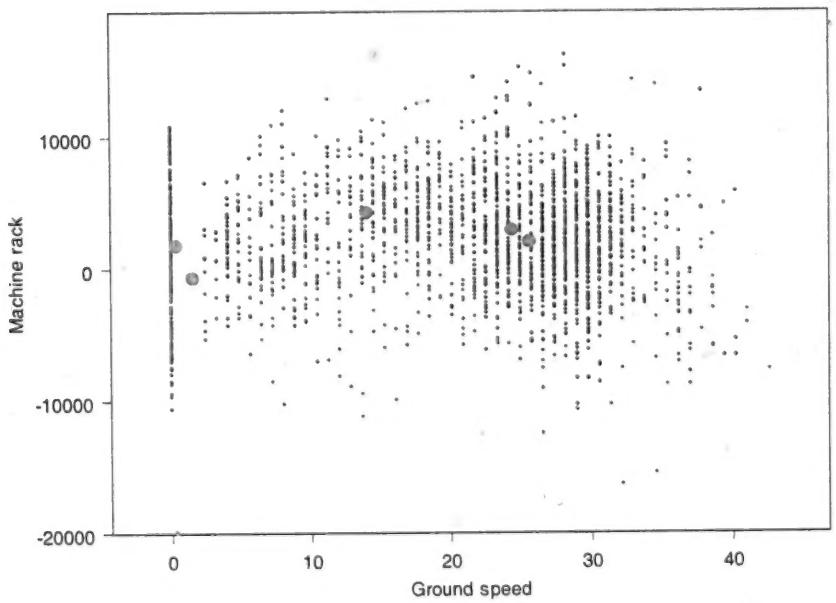


Figure 4.3 Scatter plot for the pair of inputs x3 (machine rack) vs. x1 (ground speed)

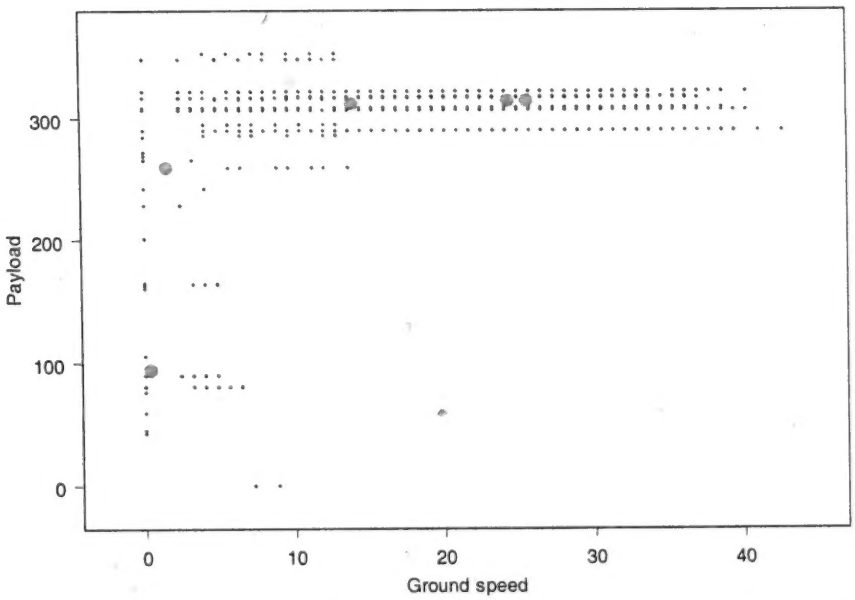


Figure 4.4 Scatter plot for the pair of inputs x4 (payload) vs. x1 (ground speed)

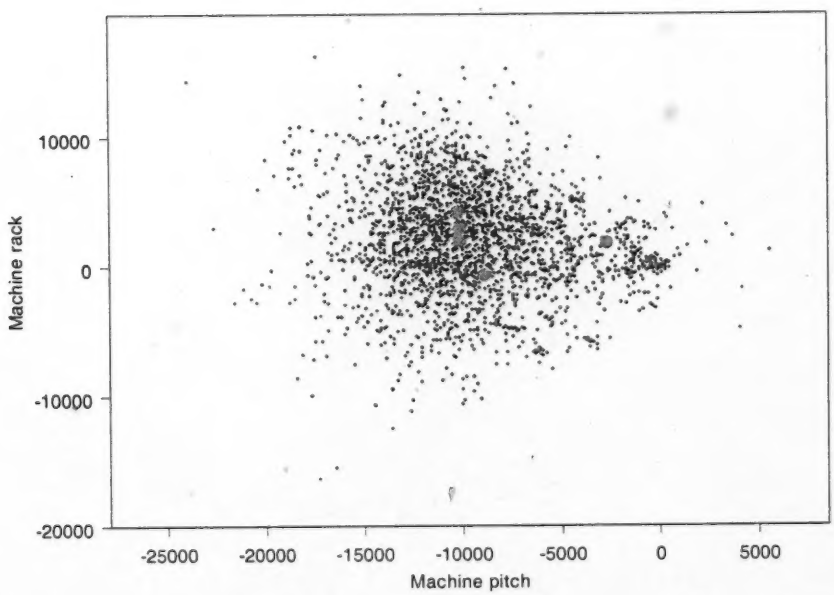


Figure 4.5 Scatter plot for the pair of inputs x3 (machine rack) vs. x2 (machine pitch)

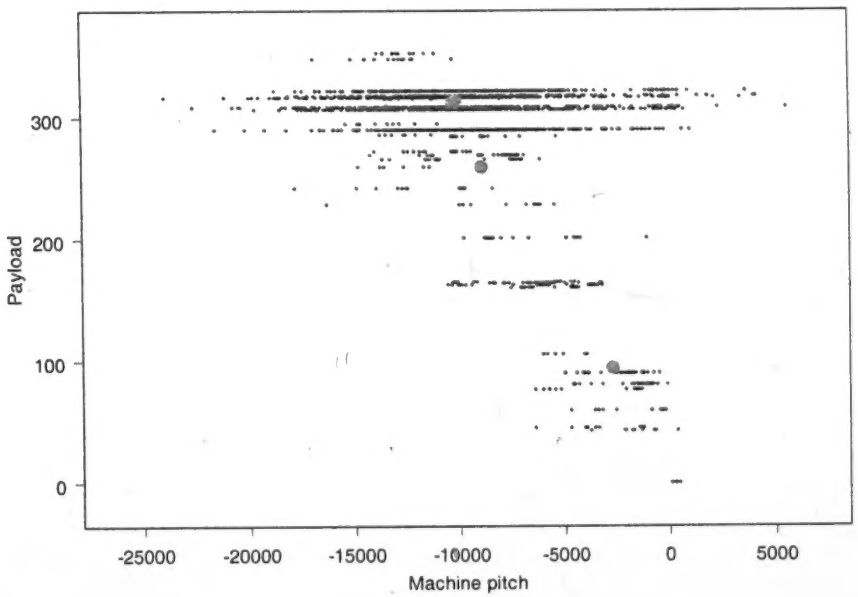


Figure 4.6 Scatter plot for the pair of inputs x4 (payload) vs. x2 (machine pitch)

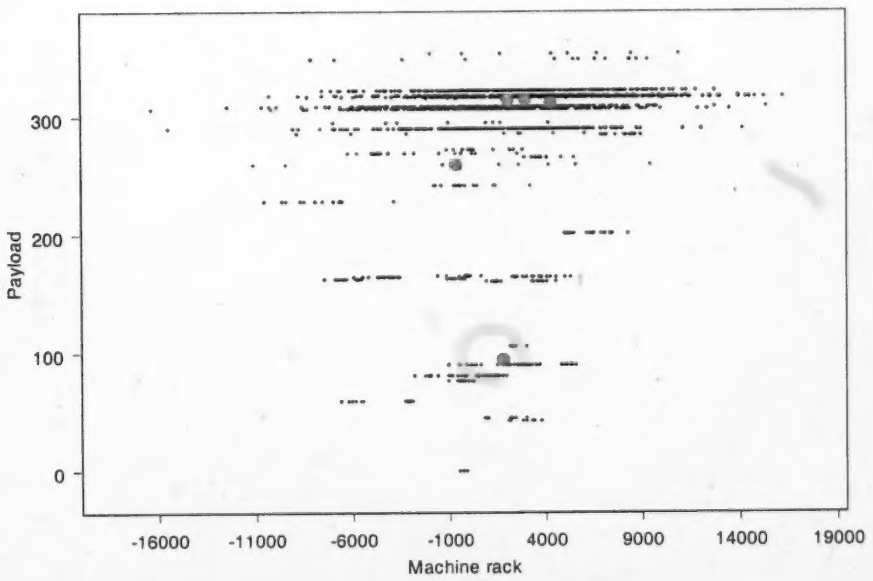


Figure 4.7 Scatter plot for the pair of inputs x4 (payload) vs. x3 (machine rack)

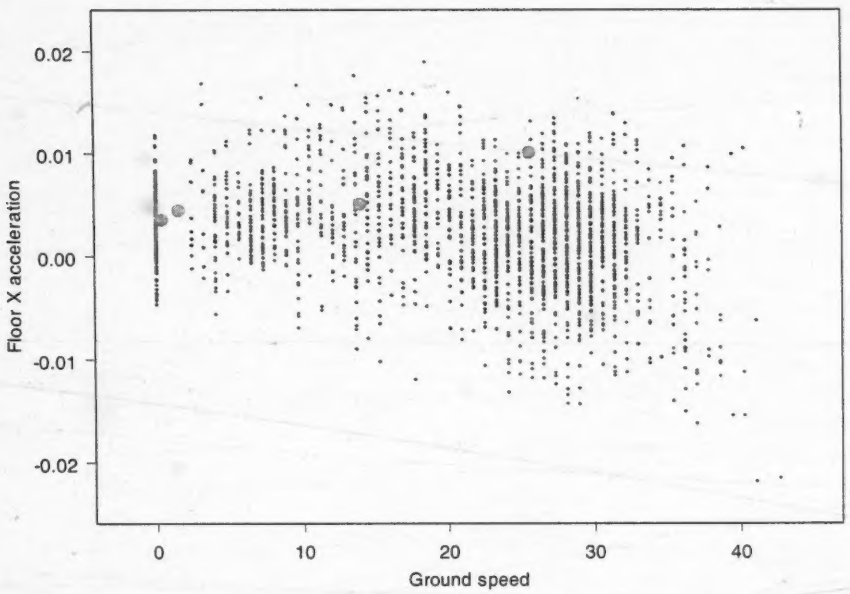


Figure 4.8 Scatter plot for the pair of output y1 (floor X acceleration) vs. input x1 (ground speed)

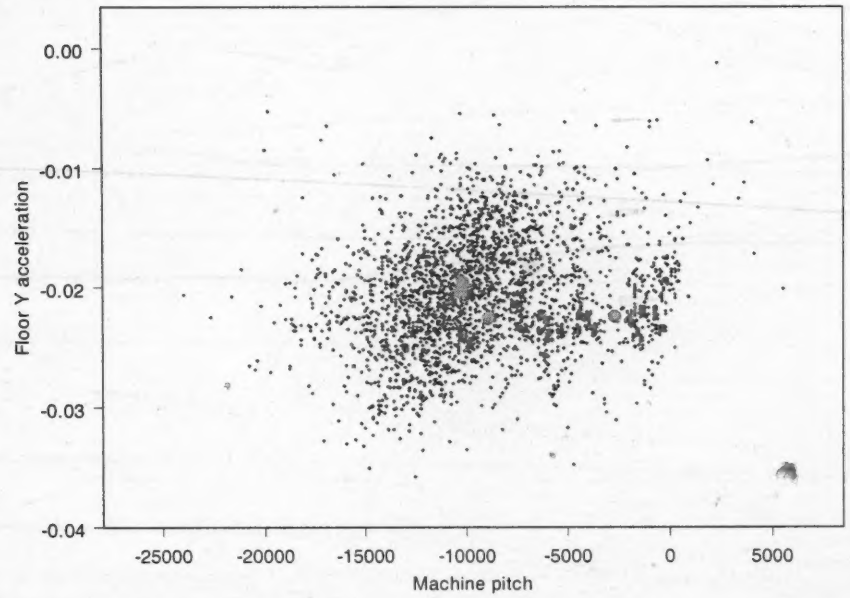


Figure 4.9 Scatter plot for the pair of output y2 (floor Y acceleration) vs. input x2 (machine pitch)

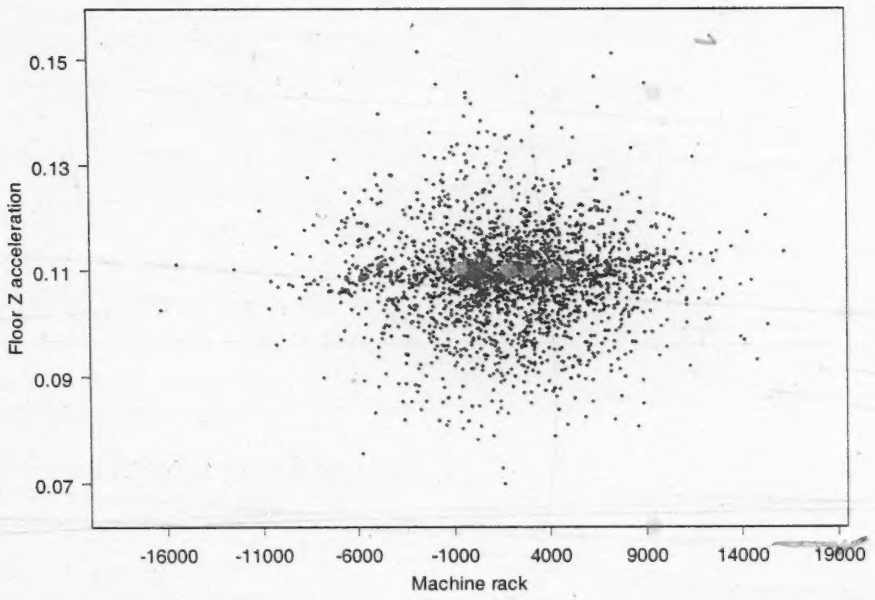


Figure 4.10 Scatter plot for the pair of output y3 (floor Z acceleration) vs. input x3 (machine rack)

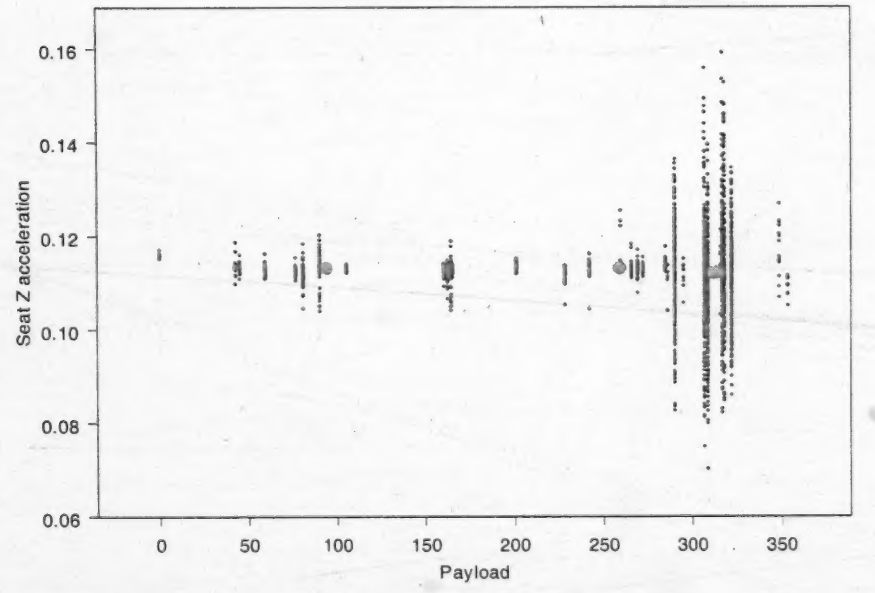


Figure 4.11 Scatter plot for the pair of output y6 (seat Z acceleration) vs. input x4 (payload)

Table 4.1 illustrates by an example the effect of clustering on standard deviation of the outputs. In this table, we can compare the standard deviation for each output in the entire dataset before clustering and in each cluster separately after applying the FCM (Fuzzy-C-Means) clustering algorithm [12]. If the dataset is divided, for example, into 8 clusters, some of them are placed in the areas of higher data concentration (about twice as lower standard deviation for almost all outputs in clusters 2, 4, and 5), however in other clusters standard deviation remains at the same level or even increases.

Table 4.1 Effect of clustering on standard deviation of the outputs

	y1	y2	y3	y4	y5	y6
Dataset	0.004428	0.004328	0.007906	0.004649	0.004641	0.008242
Cluster 1	0.00497	0.006083	0.006813	0.005685	0.005794	0.006933
Cluster 2	0.002452	0.003834	0.003251	0.002754	0.003966	0.003437
Cluster 3	0.003656	0.004298	0.008949	0.004156	0.004656	0.009535
Cluster 4	0.002475	0.003158	0.011188	0.002671	0.003872	0.011621
Cluster 5	0.002348	0.001822	0.001932	0.00197	0.001959	0.001717
Cluster 6	0.003329	0.004477	0.011846	0.003904	0.004757	0.012841
Cluster 7	0.004869	0.004213	0.011115	0.005348	0.004714	0.011444
Cluster 8	0.004485	0.003889	0.007784	0.004489	0.004371	0.007848

Presumably, in clusters with lower standard deviation, the local models would perform more precisely than when modeling the entire dataset together, therefore improving the total performance.

Table 4.2 shows as an example the cluster prototypes placed by the FCM algorithm after it was executed to create 5 clusters from the whole working dataset, including 8 inputs and 6 outputs. Two-dimensional projections of these prototypes are shown on the plots Figure 4.2, Figure 4.3, Figure 4.4, Figure 4.5, Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9, Figure 4.10, Figure 4.11.

Table 4.2 Prototypes of 5 clusters after performing FCM algorithm

Data inputs and outputs	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Input 1: Ground speed of the truck	0.359792	25.62023	24.36731	1.530682	13.99153
Input 2: Machine pitch	-2703.08	-10171	-10143	-8905.45	-10258.1
Input 3: Machine rack	1820.897	2051.666	2952.282	-636.616	4277.753
Input 4: Payload weight	94.03862	314.0263	314.3103	259.2187	311.9121
Input 5: Pressure in susp. cyl. LTF	4550.607	6419.179	6271.033	5253.98	5941.772
Input 6: Pressure in susp. cyl. LTR	4991.699	10478.85	9866.401	10025.01	8931.958
Input 7: Pressure in susp. cyl. RTF	3565.107	5916.262	6022.57	5265.174	6314.91
Input 8: Pressure in susp. cyl. RTR	5827.096	12027.6	12570.22	9399.59	13582.85
Output 1: Floor X acceleration	0.003538	0.002211	0.002873	0.004471	0.00509
Output 2: Floor Y acceleration	-0.02244	-0.01957	-0.01981	-0.02256	-0.0204
Output 3: Floor Z acceleration	0.110087	0.109605	0.109709	0.110406	0.109565
Output 4: Seat X acceleration	-0.01584	-0.01482	-0.01535	-0.01663	-0.01737
Output 5: Seat Y acceleration	-0.00975	-0.0123	-0.012	-0.00932	-0.01114
Output 6: Seat Z acceleration	0.113127	0.112205	0.112175	0.113046	0.112042

The six plots below (Figure 4.12, Figure 4.13, Figure 4.14, Figure 4.15, Figure 4.16, Figure 4.17) show the size of resulting five clusters and the values of each of the six outputs in these clusters after the entire dataset was allocated to these clusters (taking both inputs and outputs into consideration when computing the distances). Cluster boundaries showing their size are inserted in the plots. It can be seen that clusters 2 and 4 represent the most stable parts of the dataset, while clusters 1, 3, and 5 embrace the parts with the most variations in the data. Increasing the number of clusters does not help to further separate the heterogeneous part of the dataset: the different prototypes become practically identical (such as the prototypes of clusters 2 and 3 in Table 4.2). As for the prototypes of clusters 2 and 4 (more homogeneous part), they are quite different, although it is not visible on the plots.

To understand how the data are grouped and decide on the number of clusters that would be logical for this particular dataset, Principal Component Analysis (PCA) was performed on the dataset. After the principal components were obtained, we could see that the first component explains 59% of the variance and the second component explains 33%, i.e. two of them together explain almost all the variance (92%). The two plots below represent the first and second principal components, after PCA was performed on 8 inputs (Figure 4.18) and on the entire dataset, i.e. 8 inputs and 6 outputs (Figure 4.19). It is difficult to determine from these plots how many clusters would be optimal for the partition of the data. We can probably see one cluster in the corner separate from the major data cloud. Plotting the other more or less significant pairs of components (e.g. 1–3, 1–4, 2–3), as well as 3D plot (components 1–2–3) does not add any clarity into deciding on the optimal number of clusters, since these plots also produce a data cloud without clearly visible clusters.

We can conclude that data visualization techniques considered here haven't provided any clear suggestions on the number of clusters to choose for the FCM algorithm. While building the clustering-based data analysis models, we experimented within a reasonable number of clusters, while modifying the other model parameters for each number of clusters, to obtain the results that could be compared to each other.

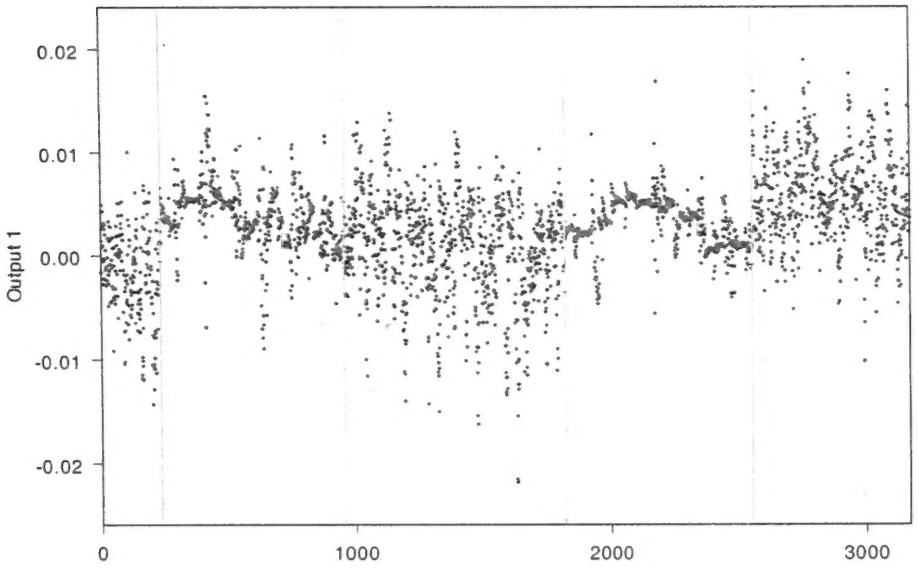


Figure 4.12 Output y1 (Floor X acceleration) values sorted by 5 clusters

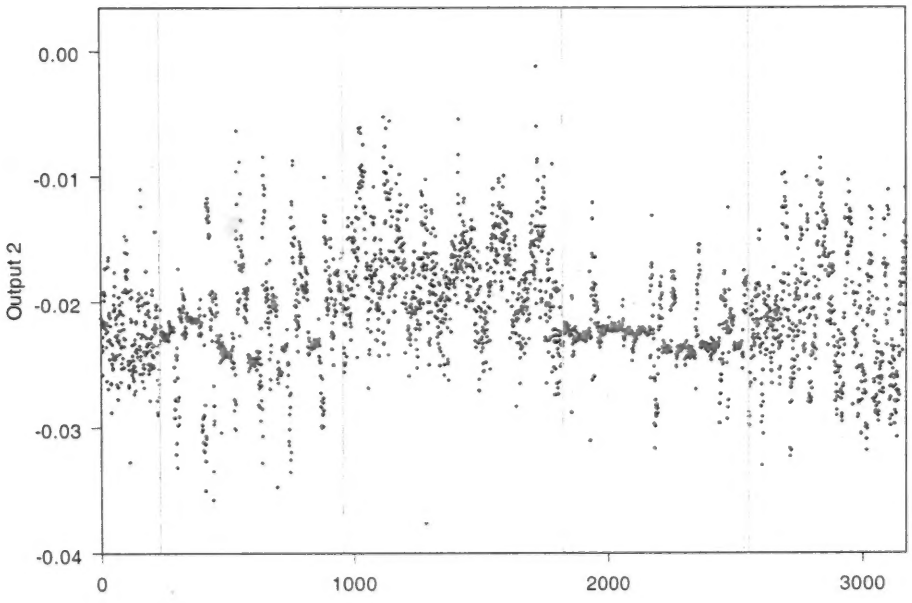


Figure 4.13 Output y2 (Floor Y acceleration) values sorted by 5 clusters

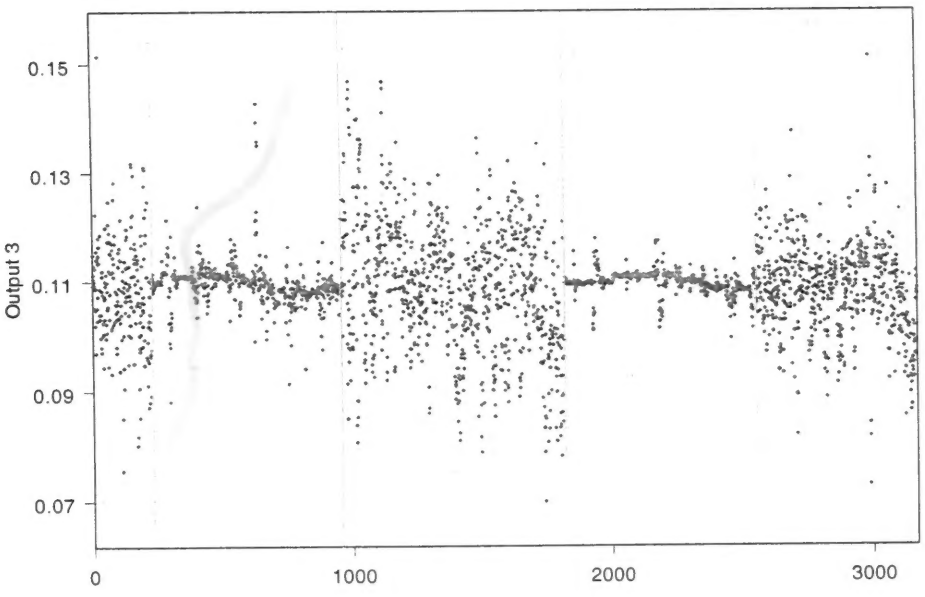


Figure 4.14 Output y3 (Floor Z acceleration) values sorted by 5 clusters

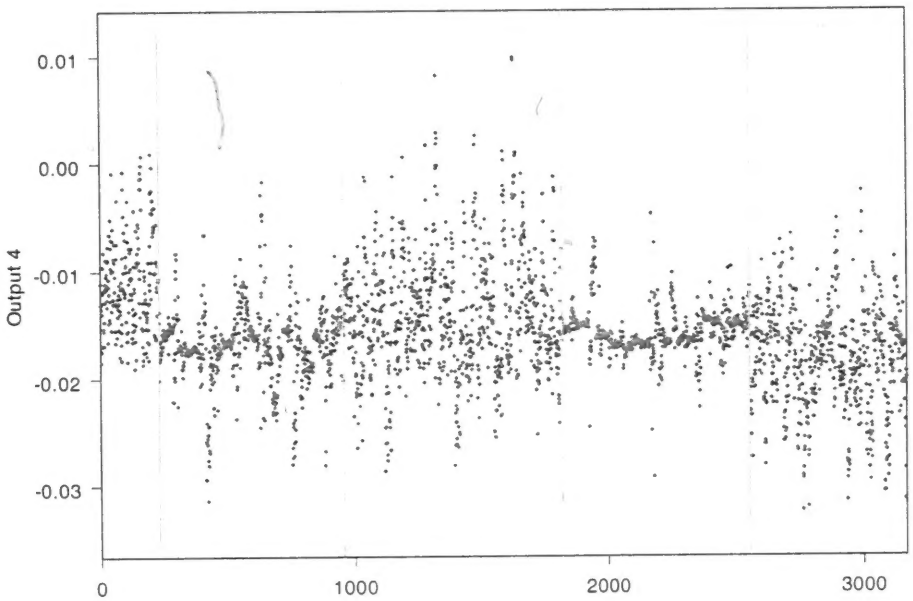


Figure 4.15 Output y4 (Seat X acceleration) values sorted by 5 clusters

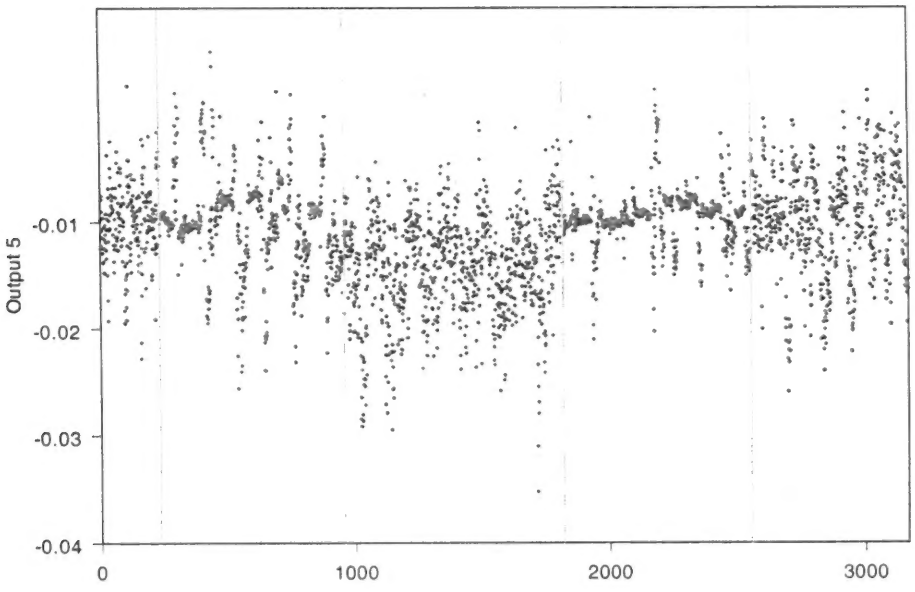


Figure 4.16 Output y5 (Seat Y acceleration) values sorted by 5 clusters

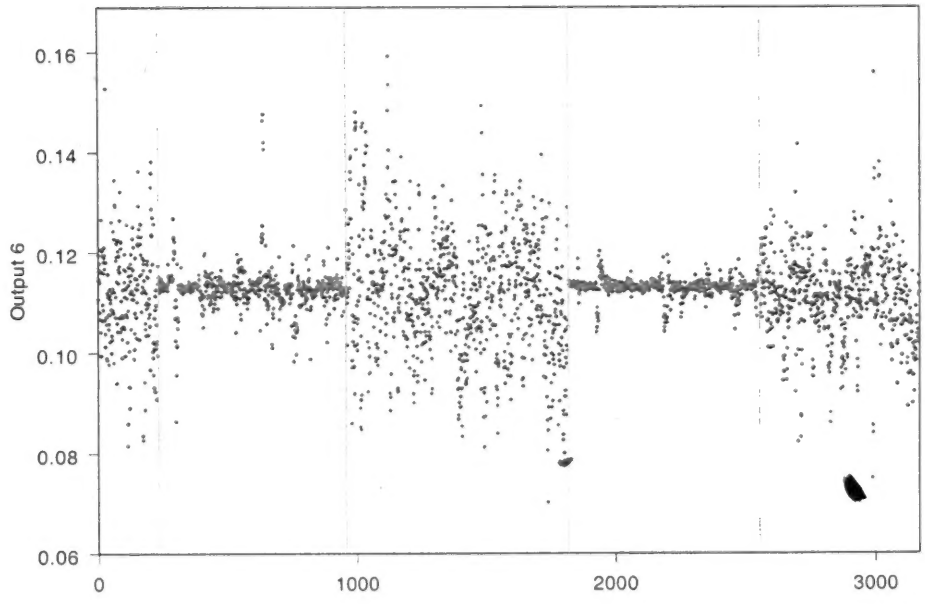


Figure 4.17 Output y6 (Seat Z acceleration) values sorted by 5 clusters

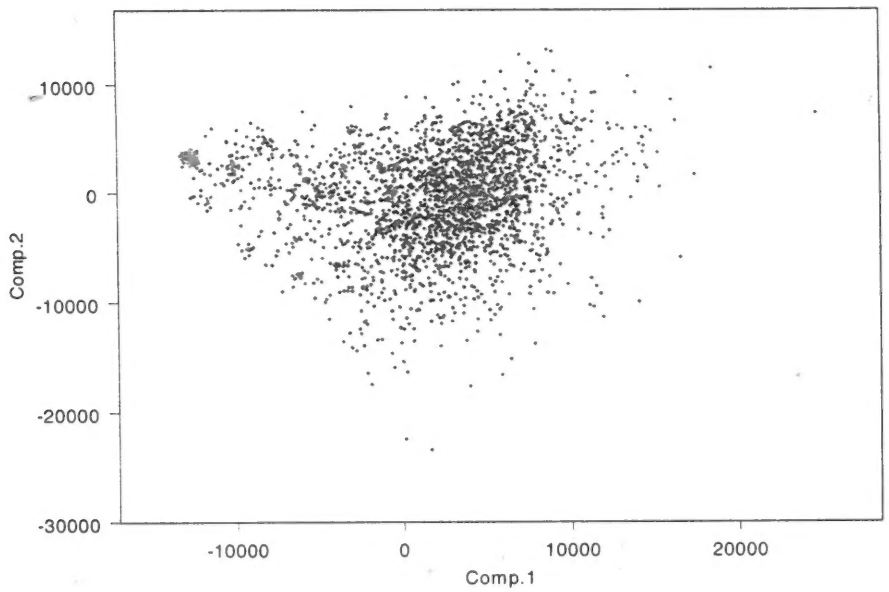


Figure 4.18 Plot of two first components after PCA performed on 8 inputs of the dataset

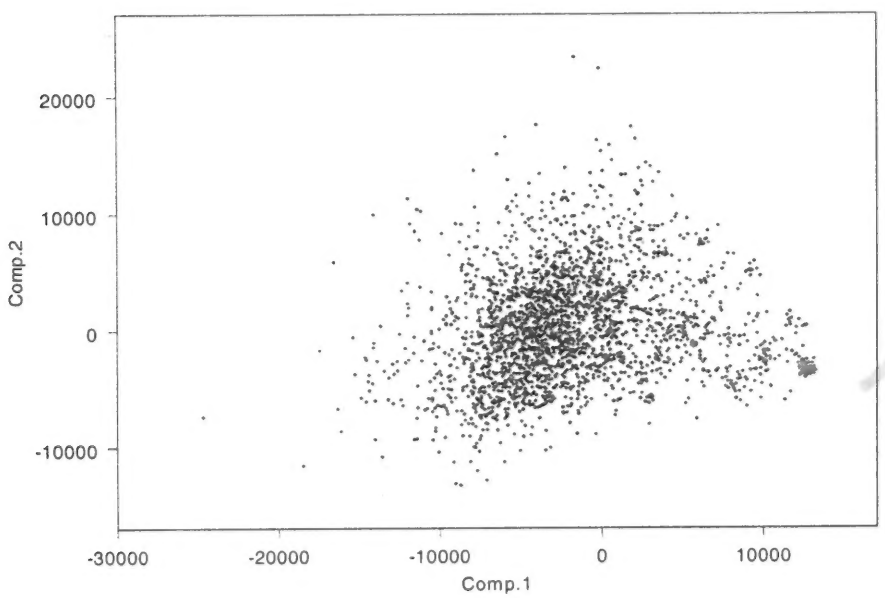


Figure 4.19 Plot of two first components after PCA performed on all columns (8 inputs and 6 outputs) of the dataset

4.4. Local linear models

4.4.1. Least square solution for a linear regression

The basic model of a simple (i.e. with one independent variable) linear regression is written as follows [13]:

$$y = b_0 + b_1 x \tag{4.8}$$

where

- x is an independent variable
- y is a dependent variable
- b_0 and b_1 are the parameters that have to be estimated

An error term indicating the absence of an exact linear relationship between x and y can be expressed from (4.8) as a difference between the dependent variable and the output modeled by the linear regression:

$$e = y - (b_0 + b_1 x) \tag{4.9}$$

Finding the least square solution for a linear regression requires minimizing of the objective function that represents the sum of squared distances between real data points and corresponding approximations of these data points as they are computed by the linear model [13]:

$$L = \sum_{i=1}^{N} e_i^{\,2} = \sum_{i=1}^{N} \left(y_i - (b_0 + b_1 x_i)\right)^2 \tag{4.10}$$

where N is the number of data points in a dataset estimated by the linear model.

In the case of n-dimensional independent variable demanding multiple regression

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n + e \tag{4.11}$$

the objective function expression (4.10) looks as follows:

$$L = \sum_{i=1}^{N} \left(y_i - (b_0 + b_1 x_{i1} + b_2 x_{i2} + ... + b_n x_{in}) \right)^2 \qquad (4.12)$$

With matrix algebra notation, in general, the regression model can be written in the following form [14]:

$$y = Xb + e \qquad (4.13)$$

where, for multiple regression n-dimensional input space:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1n} \\ 1 & x_{21} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ 1 & x_{N1} & \dots & x_{Nn} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \\ \dots \\ b_n \end{bmatrix} \qquad \mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \dots \\ e_N \end{bmatrix}$$

The first column of matrix **X** is added to produce the b_0 term when the **X**×**b** matrix multiplication is performed [14].

Error vector **e** can be expressed from (4.13) as

$$\mathbf{e} = \mathbf{y} - \mathbf{Xb} \qquad (4.14)$$

The objective function (4.10) in matrix notation is expressed as

$$L = \mathbf{e}^T \mathbf{e} \qquad (4.15)$$

Substituting **e** with expression (4.14) and opening the brackets, we obtain [15]:

$$\begin{aligned} L &= (\mathbf{y} - \mathbf{Xb})^T (\mathbf{y} - \mathbf{Xb}) = \\ &= \mathbf{y}^T \mathbf{y} - \mathbf{b}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{Xb} + \mathbf{X}^T \mathbf{X} \mathbf{b}^T \mathbf{b} = \\ &= \mathbf{y}^T \mathbf{y} - 2\mathbf{b}(\mathbf{X}^T \mathbf{y}) + \mathbf{b}^T (\mathbf{X}^T \mathbf{X})\, \mathbf{b} \end{aligned} \qquad (4.16)$$

To find a solution for **b** that minimizes the objective function L, the expression (4.16) is differentiated with respect to vector **b** and the result is set equal to the zero vector [15]:

$$\nabla_b L = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{0} \tag{4.17}$$

After moving $-2\mathbf{X}^T\mathbf{y}$ to the right hand side of (4.17) and dividing both sides by 2, (4.17) becomes

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y} \tag{4.18}$$

After multiplying both sides of (4.18) by the inverse matrix $(\mathbf{X}^T\mathbf{X})^{-1}$, we obtain the expression for **b**, which represents least square solution for (4.17):

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{4.19}$$

The solution applies to any multiple linear regression regardless of the number of regressors [13]. In other words, the least square solution (4.19) is universal for n-dimensional independent variable, thus producing n+1 parameters: $b_0, b_1, \ldots, b_n$.

In the case of m-dimensional output space (m dependent variables), for each of them there will be a separate solution, vector $\mathbf{b}_i$ ($i = 1, 2, \ldots, m$), thus creating $(n+1)\times m$ matrix **B**.

4.4.2. Theoretical base of the local linear regressions model

Linear regression is one of the simplest data analysis tools and it can be always tried as a reference point for all the other models and methods. However, drawing a single straight line throughout an entire dataset containing thousands of data points inevitably produces substantial discrepancies between the real data and the linear model outputs.

The hypothesis is that, if the data are separated into several different groups (clusters), then probably each group could be described more precisely with a separate linear model. The approach is the same that makes a non-linear function be described more precisely with several piece-wise linear approximations with different parameters for different parts of its argument, rather than with a single linear regression throughout the whole dataset:

$$R_i: \quad \text{If } x_1 \text{ is } A_1, \text{ and } \ldots, \text{ and } x_n \text{ is } A_n \quad \text{then } y = p_0 + p_1 x_1 + \ldots + p_n x_n \tag{4.20}$$

This method explores the performance of the model based on the data partition into a number of clusters (c). In each cluster, there is a center (prototype), around which the data points group together. Each data point is assigned to a single specific cluster based on the criteria of proximity of this particular data point to the prototype of this particular cluster. In each cluster, its local implication R_i (where i = 1, 2 ,..., c) is valid, representing that particular cluster as a local model adjusted for that specific group of data.

Output space is m-dimensional throughout all (c) clusters. We can assume that in a cluster i there are N_i data points (each of m independent output variables have N_i observed values corresponding to each of N_i samples of input space of that cluster). With this assumption, the linear model value of a specific output y_j for a data point $\mathbf{x}_k$ centered around the cluster prototype $\mathbf{v}_i$ of this particular local model is written as follows:

$$y^i_{kj} = p^i_{0j} + p^i_{1j}(x^i_{k1} - v^i_1) + \ldots + p^i_{nj}(x^i_{kn} - v^i_n) \tag{4.21}$$

The parameters $p^i_{0j}, p^i_{1j}, \ldots, p^i_{nj}$ (j = 1, 2, ..., m) for each of m variables of the output space of that particular local model i can be obtained using the least squares method (4.19):

$$\mathbf{p}_j^i = (\mathbf{X}_i^T \mathbf{X}_i)^{-1} \mathbf{X}_i^T \mathbf{y}_j^i \tag{4.22}$$

where

$\mathbf{X}_i$ is a working matrix of dimension $(N_i \times (n+1))$ consisting of the input space matrix of that particular cluster i (with all the data points centered around the cluster prototype $\mathbf{v}_i$), and an additional first column with all elements equal to 1, to compute $\mathbf{p}_0$ [14]:

$$\mathbf{X}_i = \begin{bmatrix} 1 & x^i_{11} - v^i_1 & \dots & x^i_{1n} - v^i_n \\ 1 & x^i_{21} - v^i_1 & \dots & x^i_{2n} - v^i_n \\ \dots & \dots & \dots & \dots \\ 1 & x^i_{N_i 1} - v^i_1 & \dots & x^i_{N_i n} - v^i_n \end{bmatrix} \tag{4.23}$$

Since the analyzed dataset has several output variables y_j (j= 1, 2 ,..., m), therefore the vector $\mathbf{y}^i_j$ represents the target output j of the cluster i, where $target^i_{kj}$ is the target value of this particular output variable y_j for the data point $\mathbf{x}_k$ of cluster i (the output already existing in the dataset and taken from there):

$$\mathbf{y}^i_j = \begin{bmatrix} target^i_{1j} \\ target^i_{2j} \\ \dots \\ target^i_{N_i j} \end{bmatrix} \tag{4.24}$$

and vector of parameters to be defined for cluster i and output variable j:

$$\mathbf{p}^i_j = \begin{bmatrix} p^i_{0j} \\ p^i_{1j} \\ \dots \\ p^i_{nj} \end{bmatrix} \tag{4.25}$$

where

- n is the number of inputs (dimensionality of the input space)
- m is the number of outputs in the dataset
- N_i is the number of data points in the cluster i

The formula (4.22) is used to find the parameters for each of the m outputs separately.

4.4.3. Applying the model to the analyzed dataset

Each of the 8 input and 6 output variables of the analyzed dataset were normalized into the interval [0,1] and then the whole dataset containing 3170 data points representing 6 working cycles was divided into the training and testing parts. The model was trained on the first two thirds of the dataset (2155 data points representing first 4 working cycles) and then tested on the remaining one third (1015 data points representing last 2 working cycles).

The training part then was clustered into groups of data points. Clustering of the data was performed with the Fuzzy-C-Means (FCM) algorithm [12], in two different ways: for the input variables only, and for the inputs and outputs together. The FCM performance index was computed as a summation of all squared Euclidian distances (between each data point and each cluster prototype) weighted by their respective membership coefficients from the partition matrix. The number of clusters was varied from 2 to 8. Allocation of the data points to the obtained clusters was performed by two different methods:

1. Choosing the maximum membership coefficients from the partition matrix that was obtained as result of executing the FCM algorithm. For example, if a fragment of the partition matrix would follow

Cluster\datapoint	...	# 523	# 524	...
1	...	0.234	0.345	...
2	...	0.511	0.262	...
3	...	0.255	0.393	...

   Then, in this example, data point #523 would be allocated to cluster 2 (since 0.511 is the maximum membership coefficient in the column 523), data point #524 to cluster 3 (maximum is 0.393), and so on.

2. By defining the receptive fields of each cluster as Gaussian functions of n-dimensional hyper-bell shape surrounding the cluster prototypes, with their parameters computed separately for

each cluster. Here, the belongingness of the data point $\mathbf{x}_k$ to this or that specific cluster is defined by a maximum value b_i among all Gaussian functions $b_1, b_2, \ldots, b_i, \ldots, b_c$ (where c is the number of clusters) for that specific data point $\mathbf{x}_k$:

If, for the data point $\mathbf{x}_k$, $b_i = \max (b_1, b_2, \ldots, b_i, \ldots, b_c)$, then the data point $\mathbf{x}_k$ is allocated to the cluster (i). The values of the Gaussian functions are computed as follows:

$$b_i = e^{\frac{-\|\mathbf{x}_k - \mathbf{v}_i\|^2}{\sigma_i^2}} \tag{4.26}$$

where

- $\| \mathbf{x}_k - \mathbf{v}_i \|$ is the Euclidian distance between data point $\mathbf{x}_k$ and prototype $\mathbf{v}_i$,
- σ_i is standard deviation of the Gaussian function b_i (in a sense, the radius of cluster i), computed by the P-nearest neighbors method [3]:

$$\sigma_i = \sqrt{\frac{1}{P}\sum_{j=1}^{P} \| \mathbf{v}_i - \mathbf{v}_j \|} \tag{4.27}$$

where

- P is the number of nearest neighbors considered for the prototype $\mathbf{v}_i$,
- $\| \mathbf{v}_i - \mathbf{v}_j \|$ is the Euclidian distance between prototype $\mathbf{v}_i$ and prototype $\mathbf{v}_j$ (one of the P nearest neighbors of prototype $\mathbf{v}_i$)

With each of the two described methods of data allocation into obtained clusters, the identical set of experiments was conducted. In the second method, P = c–1, which means that, for each cluster, all the other clusters are considered nearest neighbors.

After completing clustering, for each cluster i the local linear model was applied in the following way. The working matrix $\mathbf{X}_i$ was built from the input variables centered around the cluster prototype $\mathbf{v}_i$ (not normalized) as in (4.23), and for that working matrix, linear regression

parameters $\mathbf{p}_j^i$ were computed as in (4.22). Then, for each data point, the outputs of the model were computed as in (4.21) and compared to the target outputs, to evaluate the error. The performance index for the training part was computed as RMSE – Root Mean Squared Error (3.3) throughout all m=6 outputs and all N=2155 data points of training dataset.

To test the model, each data point of the testing part of the dataset was assigned to one of the existing clusters, using the same method of data allocation for the testing part of the dataset that was used for the training part. Then the working matrix $\mathbf{X}_i$ for each cluster of the testing dataset was built in the same way as in (4.23), and parameters $\mathbf{p}_j^i$ obtained from the training part were applied to each testing working matrix $\mathbf{X}_i$, to obtain the model outputs on the testing data. Again, for each testing data point, the outputs of the model were computed as in (4.21) and the performance index for the testing part was computed as RMSE (3.3) for all m=6 outputs and all N=1015 data points of testing dataset.

The experiments were conducted for the number of clusters from 2 to 8, with two different methods of computing the distances for data partition and two different methods of data allocation into obtained clusters. The results of the experiments are summarized in Table 4.3 and Table 4.4.

Table 4.3 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix).

Data partition:	by inputs only		by inputs and outputs	
# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.015471	0.018190	0.005358	**0.005820**
3	0.005405	0.006134	0.005280	0.006095
4	0.012628	0.010183	0.005218	0.006234
5	0.005434	0.006483	0.005163	0.006193
6	0.005991	0.007355	0.005115	0.006310
7	0.005894	0.007862	0.005186	0.006400
8	0.010567	0.010469	0.005142	0.006260

Table 4.4 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster).

Data partition:	by inputs only		by inputs and outputs	
# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.015471	0.018190	0.005358	**0.005820**
3	0.005358	0.005851	0.005287	0.005960
4	0.006022	0.006717	0.005195	0.006128
5	0.006488	0.006841	0.005313	0.006206
6	0.010830	0.007891	0.005209	0.006226
7	0.006996	0.007697	0.005338	0.006426
8	0.008409	0.009374	0.010994	0.010785

The best result on the testing dataset (RMSE = 0.005820, highlighted in boldface) was obtained by the model with 2 clusters created considering both inputs and outputs when computing the distances from data points to the prototypes.

For reference, linear regression computed for the training dataset, and tested with the same parameters on the testing dataset, provided the following results: training RMSE = 0.005407, testing RMSE = 0.005889.

The difference between the model performance on the outputs y1, y2, y4, and y5 from one side (acceleration dimensions X and Y), and on the outputs y3 and y6 from the other side (acceleration dimension Z), was confirmed again by separating these outputs from each other. First, the experiments were conducted on the dataset containing all 8 inputs and only the outputs y1, y2, y4, and y5. The performance results were as follows:

Model type	No clustering	2 clusters	3 clusters	4 clusters
Training RMSE	0.003597	0.003522	0.003457	0.003461
Testing RMSE	0.003917	0.003814	0.003992	0.003917

Then the same experiments were repeated on the dataset containing same inputs and only the outputs y3 and y6, with the following results:

Model type	No clustering	2 clusters	3 clusters	4 clusters
Training RMSE	0.007864	0.007863	0.007774	0.007664
Testing RMSE	0.008565	0.008613	0.008789	0.009033

Same as for the neural network models, in all these experiments, the results on the outputs y3 and y6 are always worse than for the outputs y1, y2, y4, and y5 (RMSE about twice as higher).

The cross-validation experiments (same series of experiments repeated 10 times for 10 different random data splits into training and testing parts) are summarized as average RMSE values and standard deviations in the Table 4.5 and Table 4.6.

Table 4.5 Average cross-validation results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix).

Data partition:	by inputs only				by inputs and outputs			
# of clust.	Trn mean RMSE	Std. dev. of RMSE	Tst mean RMSE	Std. dev. of RMSE	Trn mean RMSE	Std. dev. of RMSE	Tst mean RMSE	Std. dev. of RMSE
2	0.007176	0.002928	0.007436	0.003786	0.006025	0.000241	**0.006058**	0.000145
3	0.006093	0.000297	0.006158	0.000191	0.006016	0.000359	0.006138	0.000229
4	0.019636	0.026876	0.020079	0.028071	0.006051	0.000597	0.006187	0.000563
5	0.008744	0.004526	0.009184	0.005461	0.007297	0.004025	0.007457	0.004108
6	0.009258	0.005453	0.009254	0.004548	0.006736	0.002491	0.006969	0.002393
7	0.013221	0.009729	0.013170	0.009054	0.005769	0.000293	0.006089	0.000199
8	0.013349	0.008835	0.013069	0.008088	0.005821	0.000349	0.006349	0.000521

Table 4.6 Average cross-validation results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster).

Data partition:	by inputs only				by inputs and outputs			
# of clust.	Trn mean RMSE	Std. dev. of RMSE	Tst mean RMSE	Std. dev. of RMSE	Trn mean RMSE	Std. dev. of RMSE	Tst mean RMSE	Std. dev. of RMSE
2	0.007176	0.002928	0.007436	0.003786	0.006025	0.000241	**0.006058**	0.000145
3	0.009872	0.011054	0.009928	0.011000	0.005986	0.000252	0.006064	0.000135
4	0.007144	0.001378	0.007222	0.001464	0.006367	0.001063	0.006483	0.000903
5	0.007480	0.001441	0.007488	0.001483	0.014456	0.021274	0.015126	0.022720
6	0.008231	0.002078	0.008198	0.002089	0.007343	0.002923	0.007396	0.002979
7	0.012332	0.006366	0.011692	0.005805	0.006109	0.000657	0.006340	0.000643
8	0.013684	0.007076	0.013882	0.007967	0.008498	0.003530	0.008484	0.003254

The results of cross-validation experiments confirmed the results of experiments on the original dataset. It was still the model with 2 clusters created considering both inputs and outputs that showed the best average result on the testing dataset, although the average performance index after 10 experiments was slightly worse than that for the original dataset (RMSE = 0.006058 vs. 0.005820).

4.4.4. Introducing a threshold at data allocation

Many of the data points in the dataset have no clearly expressed membership in any of the clusters, but the algorithm described above assigns them to one of the clusters anyway, based on the maximum membership value.

To minimize the influence of such data points on the parameters of the linear models built for each cluster, a threshold was introduced in the process of data allocation into clusters formed by the FCM algorithm. Thus, besides looking for a maximum value of membership of a particular data point in each of the clusters, the allocation algorithm also compares this maximum membership coefficient to a threshold set to some level in the interval [0,1]. The data point is assigned to the appropriate cluster only if the membership coefficient exceeds the threshold. If it does not, the data point is not assigned to any cluster and is ignored in the process of building the model.

From an example above (paragraph 4.3.3), if a fragment of the partition matrix would look like the following and the threshold was set to be equal to 0.5

Cluster\datapoint	...	# 523	# 524	...
1	...	0.234	0.345	...
2	...	0.511	0.262	...
3	...	0.255	0.393	...

then, in this example, the data point #523 would be allocated to cluster 2, since 0.511 is the maximum membership coefficient in the column 523 and exceeds the threshold 0.5. The data point #524, however, would not be allocated to cluster 3 since the maximum membership coefficient 0.393 does not exceed the threshold. The data points like this are ignored in the process of building the linear models for the clusters.

Table 4.7 through Table 4.14 represent the results of experiments with the threshold levels from 0.4 to 0.7. With higher values of the threshold, most data points become excluded from the model, and it therefore cannot be considered a valid representation of the entire dataset.

Table 4.7 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.4

Data partition:	by inputs only		by inputs and outputs	
# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.015471	0.018190	0.005358	0.005820
3	0.007077	0.007600	0.005262	0.006149
4	0.005858	0.006099	0.003166	0.003720
5	0.005867	0.006290	No solution	
6	0.005450	0.006941	No solution	
7	No solution		No solution	
8	No solution		No solution	

Table 4.8 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.4

Data partition:	by inputs only		by inputs and outputs	
# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.005342	0.005840	0.004058	0.004592
3	0.005315	0.005738	0.003471	0.004010
4	0.005499	0.006099	0.004045	0.004539
5	0.009980	0.007089	0.003706	0.004374
6	0.07974	0.007751	0.003532	0.004637
7	0.006921	0.008034	0.003432	0.004364
8	0.019149	0.026105	0.076704	0.047722

Table 4.9 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.5

Data partition:	by inputs only		by inputs and outputs	
# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.015471	0.018190	0.005358	0.005820
3	0.005355	0.006155	No solution	
4	No solution		No solution	
5	0.005696	0.009093	No solution	
6	No solution		No solution	
7	No solution		No solution	
8	No solution		No solution	

Table 4.10 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.5

Data partition:	by inputs only		by inputs and outputs	
# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.005351	0.005806	0.003626	0.004113
3	0.005271	0.005798	0.003033	0.003566
4	0.007540	0.009232	0.003434	0.004024
5	0.011323	0.011830	0.003258	0.003855
6	0.007182	0.007193	No solution	

7	0.218919	0.237697	0.002886	0.003820
8	0.006249	0.006466	No solution	

Table 4.11 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.6

Data partition: by inputs only by inputs and outputs

# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.005480	0.005982	0.005294	0.006015
3	0.005809	0.006457	No solution	
4	No solution		No solution	
5	No solution		No solution	
6	No solution		No solution	
7	No solution		No solution	
8	No solution		No solution	

Table 4.12 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.6

Data partition: by inputs only by inputs and outputs

# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.005473	0.006064	0.003142	0.003964
3	0.005201	0.005793	0.002345	0.003096
4	0.007219	0.007130	0.002848	0.003503
5	0.028424	0.031479	0.003205	0.003496
6	0.005881	0.006503	0.002794	0.003475
7	0.027442	0.036132	No solution	
8	0.007304	0.010304	0.002718	0.031859

Table 4.13 Model results with the first data allocation method (by choosing the maximum membership coefficients from the partition matrix) and the threshold = 0.7

Data partition: by inputs only by inputs and outputs

# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.005715	0.006323	0.005081	0.005525
3	0.004896	0.005942	No solution	
4	No solution		No solution	
5	No solution		No solution	
6	No solution		No solution	
7	No solution		No solution	
8	No solution		No solution	

Table 4.14 Model results with the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster) and the threshold = 0.7

Data partition:	by inputs only		by inputs and outputs	
# of clusters	Training RMSE	Testing RMSE	Training RMSE	Testing RMSE
2	0.05121	0.006091	0.002413	0.003064
3	0.005952	0.006457	0.001774	0.002224
4	0.005397	0.006030	0.002208	0.002931
5	0.014117	0.028024	0.003548	0.005570
6	0.037138	0.055508	No solution	
7	0.011227	0.011798	No solution	
8	0.085061	0.044493	No solution	

With the high level of the threshold, most data points become excluded from consideration. For those included, the performance of the model may improve significantly (but at the expense of validity). For example, in the particular case with the threshold = 0.7, the FCM algorithm forming 3 clusters, the data partition by inputs and outputs together, and the second data allocation method (by computing the hyper-bell shape Gaussian functions for each cluster), the training RMSE is as low as 0.001774, and the testing RMSE is 0.002224:

Data points	Cluster 1	Cluster 2	Cluster 3	Excluded	RMSE
Training	477	31	96	1551	0.001774
Testing	154	6	7	847	0.002224

Here, in the training dataset, 1551 data points out of 2155 are excluded from clustering. Such a high number questions the validity of the model where most of the data are ignored. Thus, by increasing the threshold, the model improves its performance, but this is based on dealing only with the data the model finds "convenient" for itself and ignoring more and more data that might cause trouble. As a result, with increasing the threshold, the model becomes more and more selective and gradually loses its ability to deal with realistic data.

Figure 4.20 illustrates this particular experiment, plotting the output y1 data (both training and testing) allocated by FCM algorithm into cluster 1 containing most of the data points included in the model. Figure 4.21 plots the output y3 that, together with y6, usually contributes the most to the RMSE level. Since the data creating the most uncertainty are removed from the model, the target output y3 is modeled better than without threshold.

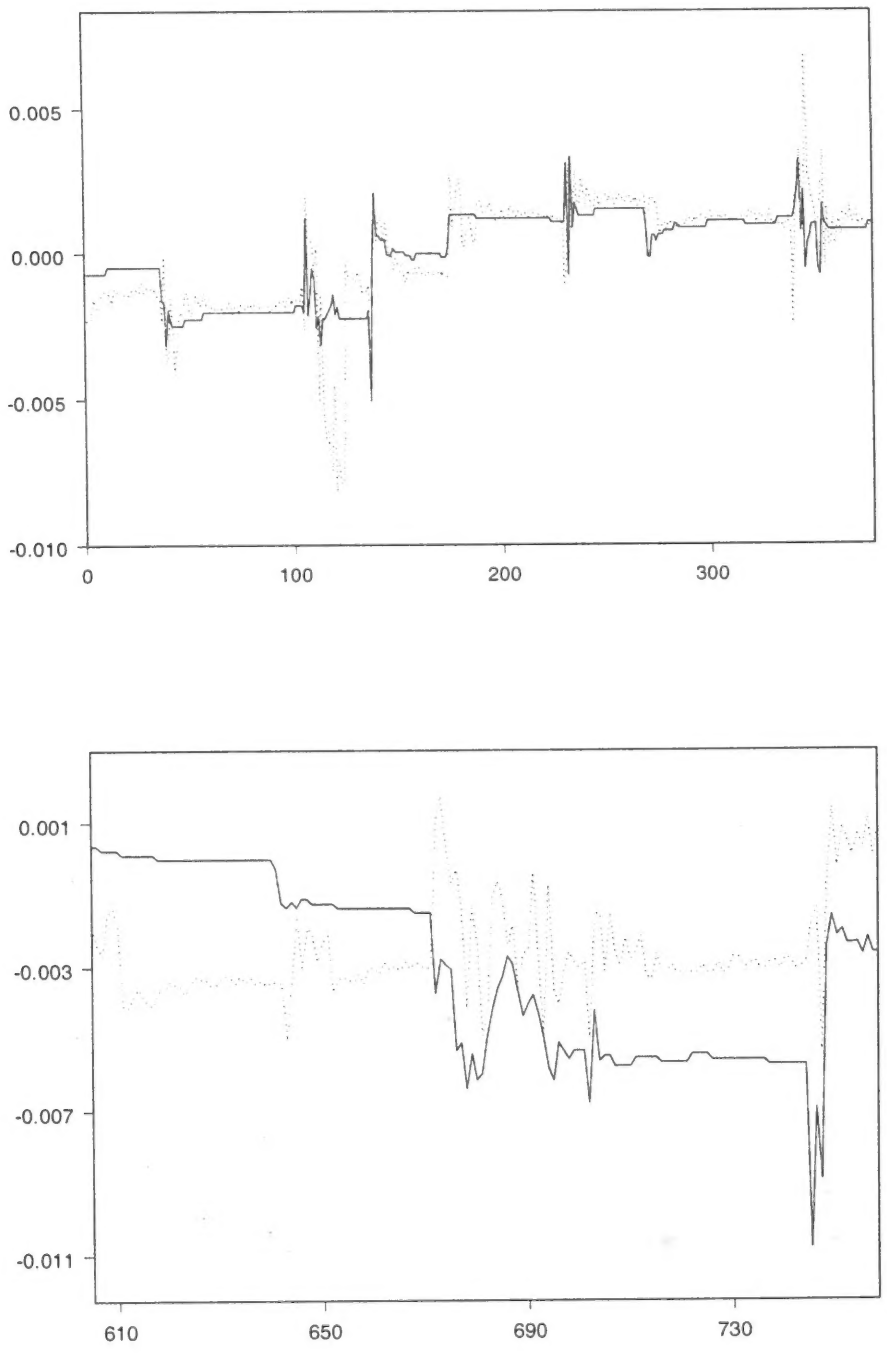


Figure 4.20 Output y1 (Floor X acceleration): dotted line – target, solid line – model output. Upper plot represents cluster 1 of the training dataset, lower plot – cluster 1 of the testing dataset.

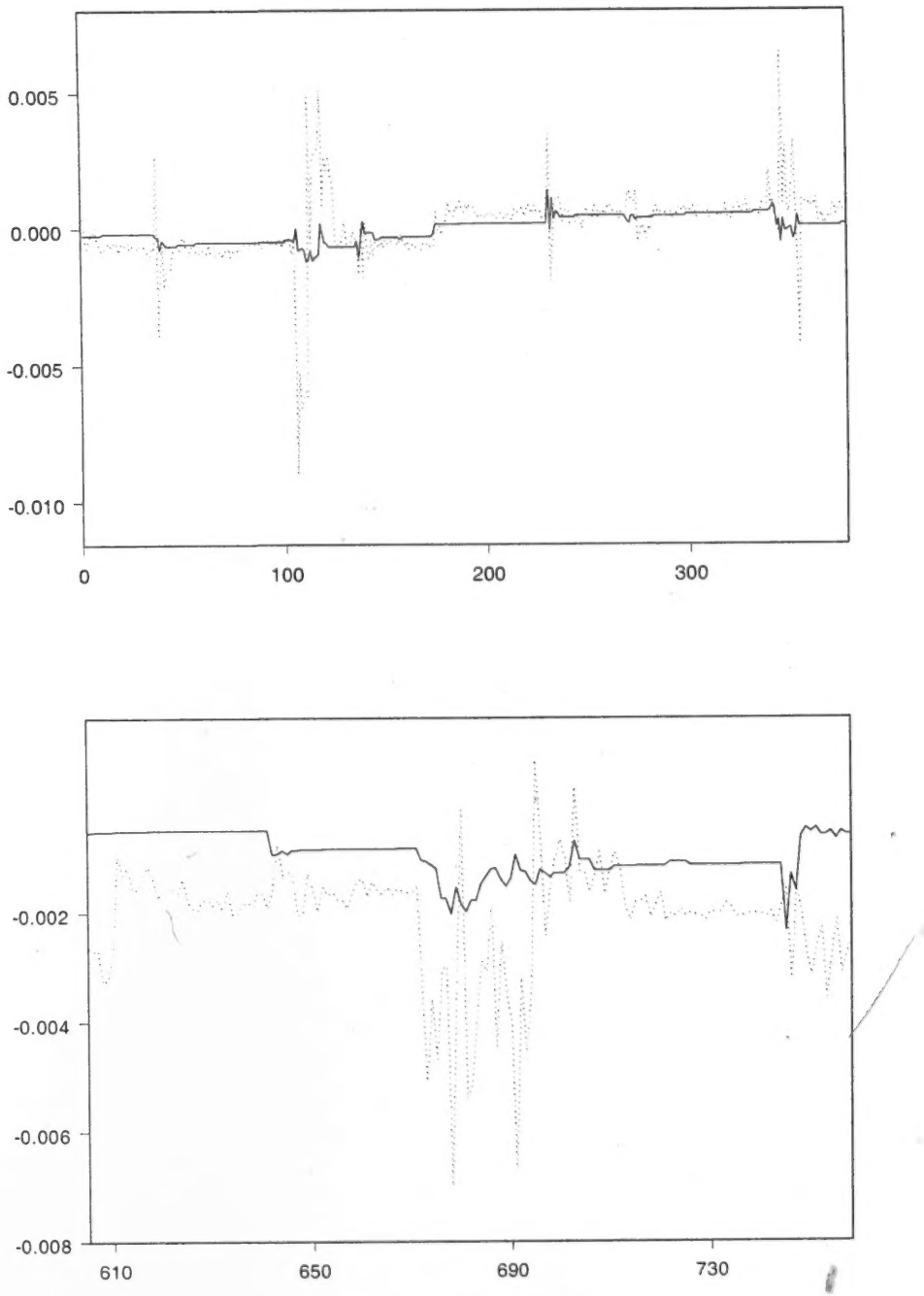


Figure 4.21 Output y3 (Floor Z acceleration): dotted line – target, solid line – model output. Upper plot represents cluster 1 of the training dataset, lower plot – cluster 1 of the testing dataset.

4.5. Takagi – Sugeno model and its application to the analyzed data

4.5.1. Theoretical base of the model

The approach was proposed by Tomohiro Takagi and Michio Sugeno in 1985 [8]. They presented a mathematical tool to build a fuzzy model of a system using fuzzy implications and reasoning. The suggested fuzzy implication is of the format [8]:

$$R: \quad \text{If } f(x_1 \text{ is } A_1, \ldots, x_k \text{ is } A_k) \text{ then } y = g(x_1, \ldots, x_k) \tag{4.28}$$

where

y	variable of the consequence whose value is inferred;
$x_1, \ldots, x_n$	variables of the premise that appear also the part of the consequence;
$A_1, \ldots, A_n$	fuzzy sets representing a fuzzy subspace where the implication R can be applied for reasoning;
f	logical function connecting the propositions in the premise
g	function that implies the value of y when $x_1, \ldots, x_k$ satisfy the premise

If in the function f only AND terms are used in the premise, and function g is adopted to be linear, then the implication R (4.28) is written as follows [8]:

$$R: \quad \text{If } x_1 \text{ is } A_1, \text{ and } \ldots, \text{ and } x_n \text{ is } A_n \quad \text{then } y = p_0 + p_1 x_1 + \ldots + p_n x_n \tag{4.29}$$

If we have several implication R_i (where $i = 1, 2, \ldots, c$) to represent the system with one single output variable, then the output y for the input $(x_1, \ldots, x_n)$ is obtained as follows:

$$y = \sum_{i=1}^{c} u_i \left(p_0^i + p_1^i x_1 + \ldots + p_n^i x_n\right) = \sum_{i=1}^{c} \left(u_i p_0^i + u_i p_1^i x_1 + \ldots + u_i p_n^i x_n\right) \tag{4.30}$$

where each coefficient $$u_i = \frac{A_1^i(x_1) \wedge A_2^i(x_2) \wedge \ldots \wedge A_c^i(x_c)}{\sum_{i=1}^{c}\left[A_1^i(x_1) \wedge A_2^i(x_2) \wedge \ldots \wedge A_c^i(x_c)\right]} \quad (4.31)$$

represents a fuzzy number in the interval [0,1] describing to which extent this particular implication R_i is valid for this particular input data point $(x_1, \ldots, x_n)$. In this approach, the entire input space is represented by (c) fuzzy subspaces, and in each of them, each implication R_i is valid with some degree of truth from 0 to 1, and the consequence function is described by (4.30).

In other words, instead of assuming that each data point is assigned to a specific cluster with membership grade 1 meaning that its membership in all other clusters is considered zero, the fuzzy approach proposes more realistic vision. From this perspective, each data point belongs to all (c) clusters and sum of all membership grades is equal to 1. Thus, to compute the output y, instead of using a specific linear model R_i for a cluster i, while ignoring all the other c–1 models, Takagi – Sugeno approach [8] uses all (c) models together, weighting their outputs with membership coefficients u_i. As a result, the output y is computed a sum of weighted outputs of all (c) models as in (4.30).

In a case of m-dimensional output space, each data point can be expressed as follows:

$$(x_{k1}, x_{k2}, \ldots, x_{kn} \rightarrow y_{k1}, y_{k2}, \ldots, y_{km}) \quad (4.32)$$

where k = 1, 2 ,…., N (number of data points in the dataset)

That means each of m independent output variables has N observed values corresponding to each of N samples of input space. The expression (4.30) for a specific output j (j = 1, 2, …, m) and data point k (k = 1, 2 ,…, N) can be written as follows:

$$y_{kj} = \sum_{i=1}^{c} u_{ik}\left(p_{0j}^i + p_{1j}^i x_{k1} + \ldots + p_{nj}^i x_{kn}\right) = \sum_{i=1}^{c} \left(u_{ik}p_{0j}^i + u_{ik}p_{1j}^i x_{k1} + \ldots + u_{ik}p_{nj}^i x_{kn}\right) =$$

$$= u_{1k}p_{0j}^1 + \ldots + u_{ck}p_{0j}^c + u_{1k}p_{1j}^1 x_{k1} + \ldots + u_{ck}p_{1j}^c x_{k1} + \ldots + u_{1k}p_{nj}^1 x_{kn} + \ldots + u_{ck}p_{nj}^c x_{kn} \quad (4.33)$$

First it is necessary to partition the space of the premise variables into a set of (c) fuzzy subspaces and to determine the membership functions of the fuzzy sets in the premises (i.e. to obtain a matrix of corresponding coefficients u_{ik}, where i = 1, 2 ,..., c; and k = 1, 2 ,..., N). To accomplish this, Takagi and Sugeno [8] used a "premise parameters identification" algorithm consisting of a series of consecutive heuristic search procedures, at each step dividing the input space into "big" and "small" areas along each dimension. In a later work [9], Sugeno and Yasukawa proposed for this purpose the FCM (Fuzzy C-Means) clustering algorithm [12]. The idea of fuzzy clustering with the FCM algorithm was also supported in [10].

In this research, the FCM algorithm is used for clustering the data. As a result of the FCM algorithm, each n-dimensional data point is fuzzily assigned to all (c) clusters. Different membership values of the data point in all these clusters produce 1 when all summed, according to the second property of the partition matrix (4.4). Therefore, each column of the resulting partition matrix can be used as a vector $\mathbf{u}_k$ (k = 1, 2 ,..., N) of coefficients u_{ik} (i= 1, 2 ,..., c) for each particular data point.

In matrix notation, (4.33) can be written as follows:

$$\mathbf{y}_j = \mathbf{X}\mathbf{p}_j \tag{4.34}$$

where

X is a working matrix of dimension (N×c(n+1)) where the first c columns represent the transposed partition matrix, and then each entry of the input space matrix is represented by c entries in the working matrix, i.e. weighted by all (c) coefficients of the partition matrix corresponding to that particular data point:

$$\mathbf{X} = \begin{bmatrix} u_{11} & \cdots & u_{c1} & u_{11}x_{11} & \cdots & u_{c1}x_{11} & \cdots & \cdots & \cdots & u_{11}x_{1n} & \cdots & u_{c1}x_{1n} \\ u_{12} & \cdots & u_{c2} & u_{12}x_{21} & \cdots & u_{c2}x_{21} & \cdots & \cdots & \cdots & u_{12}x_{2n} & \cdots & u_{c2}x_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ u_{1N} & \cdots & u_{cN} & u_{1N}x_{N1} & \cdots & u_{cN}x_{N1} & \cdots & \cdots & \cdots & u_{1N}x_{Nn} & \cdots & u_{cN}x_{Nn} \end{bmatrix} \tag{4.35}$$

$\mathbf{y}_j$ is a vector of values predicted by the model for the particular output j (j = 1, 2, ..., m):

$$\mathbf{y}_j = \begin{bmatrix} y_{1j} \\ y_{2j} \\ \cdots \\ y_{Nj} \end{bmatrix} \tag{4.36}$$

$\mathbf{p}_j$ is a vector of c(n+1) model parameters to be defined for the particular output j:

$$\mathbf{p}_j = \begin{bmatrix} p^1_{0j} \\ \cdots \\ p^c_{0j} \\ p^1_{1j} \\ \cdots \\ p^c_{1j} \\ \cdots \\ \cdots \\ \cdots \\ p^1_{nj} \\ \cdots \\ p^c_{nj} \end{bmatrix} \tag{4.37}$$

where

- n is the number of inputs (dimensionality of the input space)
- m is the number of outputs in the dataset
- N is the number of data points in the dataset
- c is the number of clusters chosen for fuzzy partition

If $\mathbf{t}_j$ is a vector of target values for output j already existing in the dataset

$$\mathbf{t}_j = \begin{bmatrix} target_{1j} \\ target_{2j} \\ \cdots \\ target_{Nj} \end{bmatrix} \tag{4.38}$$

then, according to (4.14), error vector $\mathbf{e}_j$ is expressed as:

$$\mathbf{e}_j = \mathbf{t}_j - \mathbf{y}_j = \mathbf{t}_j - \mathbf{X}\mathbf{p}_j \tag{4.39}$$

To define $\mathbf{p}_j$, we need to minimize the objective function as in (4.15) – (4.16):

$$L = \mathbf{e}_j^T\mathbf{e}_j = (\mathbf{t}_j - \mathbf{X}\mathbf{p}_j)^T(\mathbf{t}_j - \mathbf{X}\mathbf{p}_j) =$$

$$= \mathbf{t}_j^T\mathbf{t}_j - \mathbf{p}_j^T\mathbf{X}^T\mathbf{t}_j - \mathbf{t}_j^T\mathbf{X}\mathbf{p}_j + \mathbf{X}^T\mathbf{X}\mathbf{p}_j^T\mathbf{p}_j =$$

$$= \mathbf{t}_j^T\mathbf{t}_j - 2\mathbf{p}_j(\mathbf{X}^T\mathbf{t}_j) + \mathbf{p}_j^T(\mathbf{X}^T\mathbf{X})\,\mathbf{p}_j \qquad (4.40)$$

which means to differentiate it with respect to each element of $\mathbf{p}_j$, set each derivative equal to 0 and solve each equation as in (4.17) – (4.19):

$$\nabla_{\mathbf{p}_j} L = -2\mathbf{X}^T\mathbf{t}_j + 2\mathbf{X}^T\mathbf{X}\mathbf{p}_j = \mathbf{0} \qquad (4.41)$$

After moving $-2\mathbf{X}^T\mathbf{t}_j$ to the right part, dividing both parts by 2, and multiplying by the inverse matrix $(\mathbf{X}^T\mathbf{X})^{-1}$, we obtain the expression for $\mathbf{p}_j$, which represents least square solution for (4.41), i.e. all the parameters $p'_{0j}, p'_{1j}, \ldots, p'_{rj}$ necessary for predicting the model outputs y_{kj} as in (4.33):

$$\mathbf{p}_j = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}_j \qquad (4.42)$$

The formula (4.42) is used to find the parameters for each of the m outputs separately.

4.5.2. Applying the model to the analyzed dataset

Each of the 8 input and 6 output variables of the analyzed dataset was normalized into the interval [0,1] and then the whole dataset containing 3170 data points representing 6 working cycles was divided into the training and testing parts. The model was trained on the first two thirds of the dataset (2155 data points representing first 4 working cycles) and then tested on the remaining one third (1015 data points representing last 2 working cycles).

The training part was clustered with the FCM algorithm. Clustering was performed in two ways in two sets of experiments: with input variables only, and with both inputs and outputs together. The FCM algorithm stopped when it reached the termination criterion, i.e. after the iteration (i) where

the change of the FCM performance index did not exceed the FCM termination criterion set up to 0.01. The FCM performance index was computed as the sum of all squared Euclidian distances (between each data point and each cluster prototype) weighted by their respective membership coefficients from the partition matrix. The number of clusters was varied from 2 to 8.

After completing the clustering, the working matrix **X** was built from the input variables (not normalized) and the elements of partition matrix as in (4.35), and for that working matrix, linear regression parameters $\mathbf{p}_j$ were computed for each output j as in (4.42). Then, for each data point, the outputs $\mathbf{y}_j$ of the model were computed as in (4.33) and compared to the target outputs, to evaluate the error. Performance index for the training part was computed as RMSE – Root Mean Squared Error (3.3) throughout all m=6 outputs and all N=2155 data points of training dataset.

To test the model, each data point of the testing part of the dataset was fuzzily assigned to all the existing clusters, through computing the matrix of distances and the partition matrix for the testing dataset. Then the working matrix **X** for the testing dataset was built in the same way, and parameters $\mathbf{p}_j$ obtained from the training part were applied to the testing working matrix, to obtain the model outputs $\mathbf{y}_j$ on the testing data. Again, for each testing data point, the outputs y_{jk} of the model were computed as in (4.33) and the performance index for the testing part was computed as RMSE (3.3) throughout all m=6 outputs and all N=1015 data points of testing dataset.

Table 4.15 and Table 4.16 summarize the performance of Takagi – Sugeno model on the analyzed dataset. For each experiment, the tables present:

1. Number of clusters in the FCM algorithm
2. Value of the FCM performance index to which the FCM algorithm converged
3. Model RMSE on the training dataset
4. Model RMSE on the testing dataset

Table 4.15 Takagi – Sugeno model performance with only input variables used for clustering and FCM termination criterion of 0.01.

Clusters	2	3	4	5	6	7	8
FCM	231.2531	141.6776	101.6943	79.6301	64.2961	53.8281	46.4424
Training	0.006333	0.007102	0.008521	0.012586	0.027519	0.011429	0.008755
Testing	**0.006407**	0.007924	0.008857	0.014880	0.024424	0.012756	0.007851

Table 4.16 Takagi – Sugeno model performance with both inputs and outputs used for clustering and FCM termination criterion of 0.01.

Clusters	2	3	4	5	6	7	8
FCM	325.8103	206.2070	149.5211	117.4207	97.2067	83.2018	70.5937
Training	0.006636	0.033495	0.012636	0.012001	0.020003	0.023893	0.071252
Testing	0.007727	0.017651	0.013946	0.011402	0.016061	0.023449	0.039135

When the FCM termination criterion is set up to a stricter value of 0.00001, which means the convergence process continues for more iterations and produces more precise values of the cluster prototypes; the results look as presented in the Table 4.17 and Table 4.18. There is no improvement in terms of the model performance.

Table 4.17 Takagi – Sugeno model performance with only input variables used for clustering and FCM termination criterion of 0.00001.

Clusters	2	3	4	5	6	7	8
FCM	231.2510	141.6658	101.1546	79.6033	64.2961	53.8148	46.4225
Training	0.015538	0.013879	0.024871	0.009826	0.024622	0.010594	0.012098
Testing	0.017176	0.014454	0.024556	0.009251	0.024769	0.010917	0.013737

Table 4.18 Takagi – Sugeno model performance with both inputs and outputs used for clustering and FCM termination criterion of 0.00001.

Clusters	2	3	4	5	6	7	8
FCM	325.8078	205.9722	149.5137	117.4012	97.1555	83.1736	70.5709
Training	0.006724	0.008754	0.023037	0.047986	0.178764	0.134314	0.068948
Testing	0.007300	0.009668	0.023424	0.050504	0.180239	0.122394	0.064207

From Table 4.15, Table 4.16, Table 4.17, and Table 4.18, the model obtained best performance on the testing dataset (RMSE = 0.006407, marked with boldface in the Table 4.15) with the number of clusters 2, and FCM termination criterion of 0.01. Figure 4.22, Figure 4.23, and Figure 4.24 are plotted for this configuration and represent some of model outputs vs. targets for the training and

testing data. The plots are built on the fragments of the 4th operational cycle of the training dataset and of the 1st operational cycle of the testing dataset.

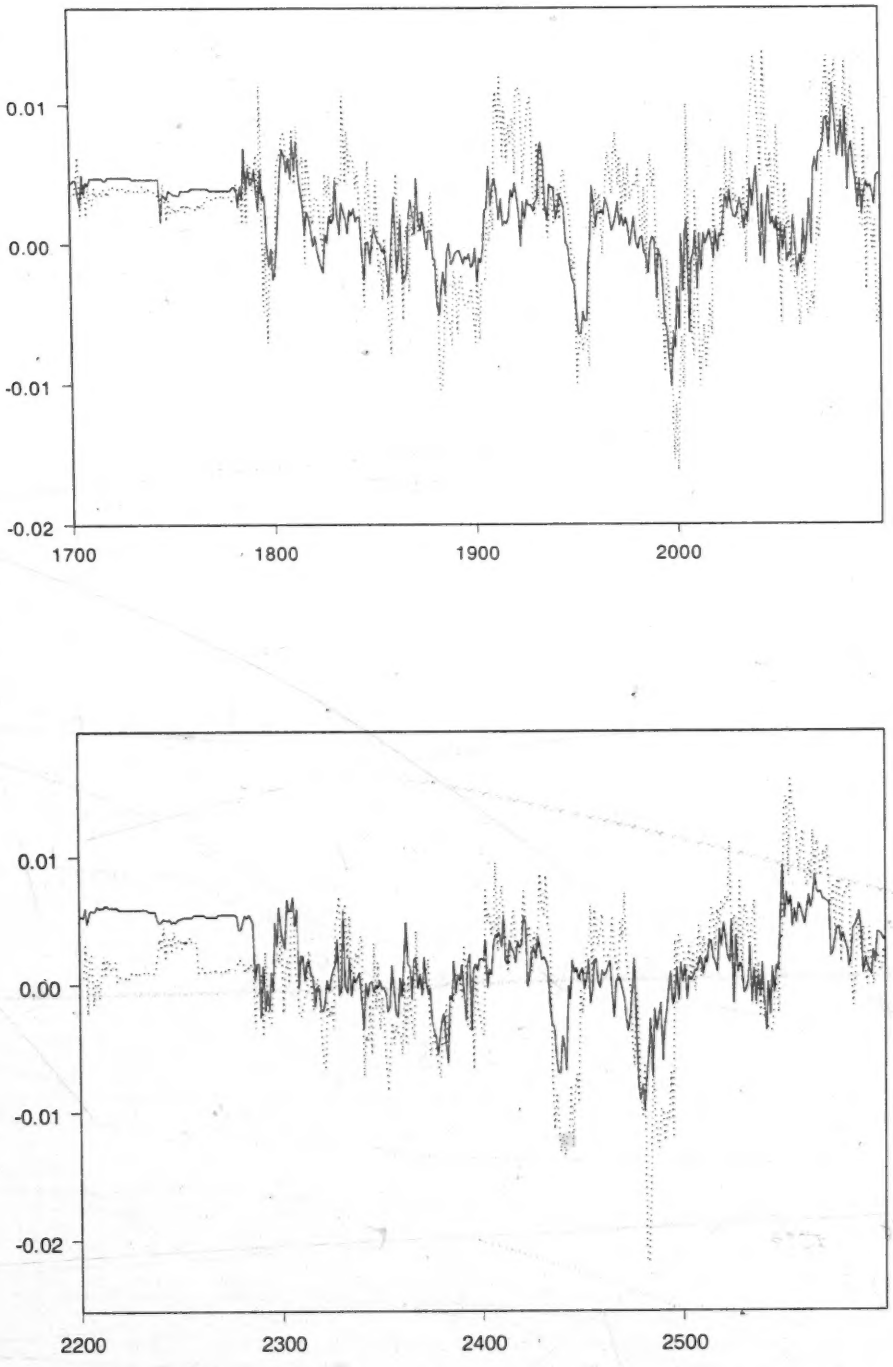


Figure 4.22 TS model, output y1 (floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data.

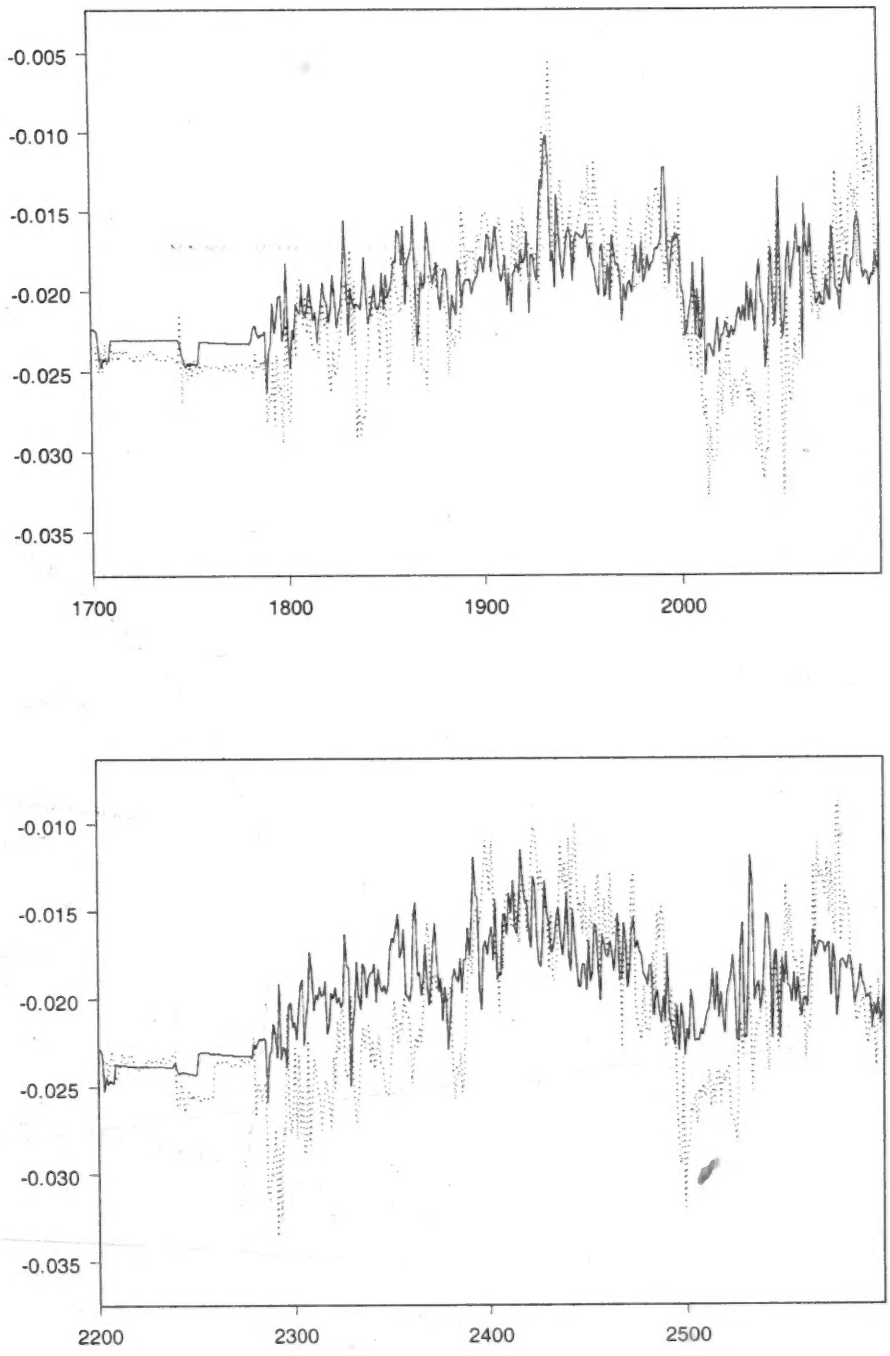


Figure 4.23 TS model, output y2 (floor Y acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data.

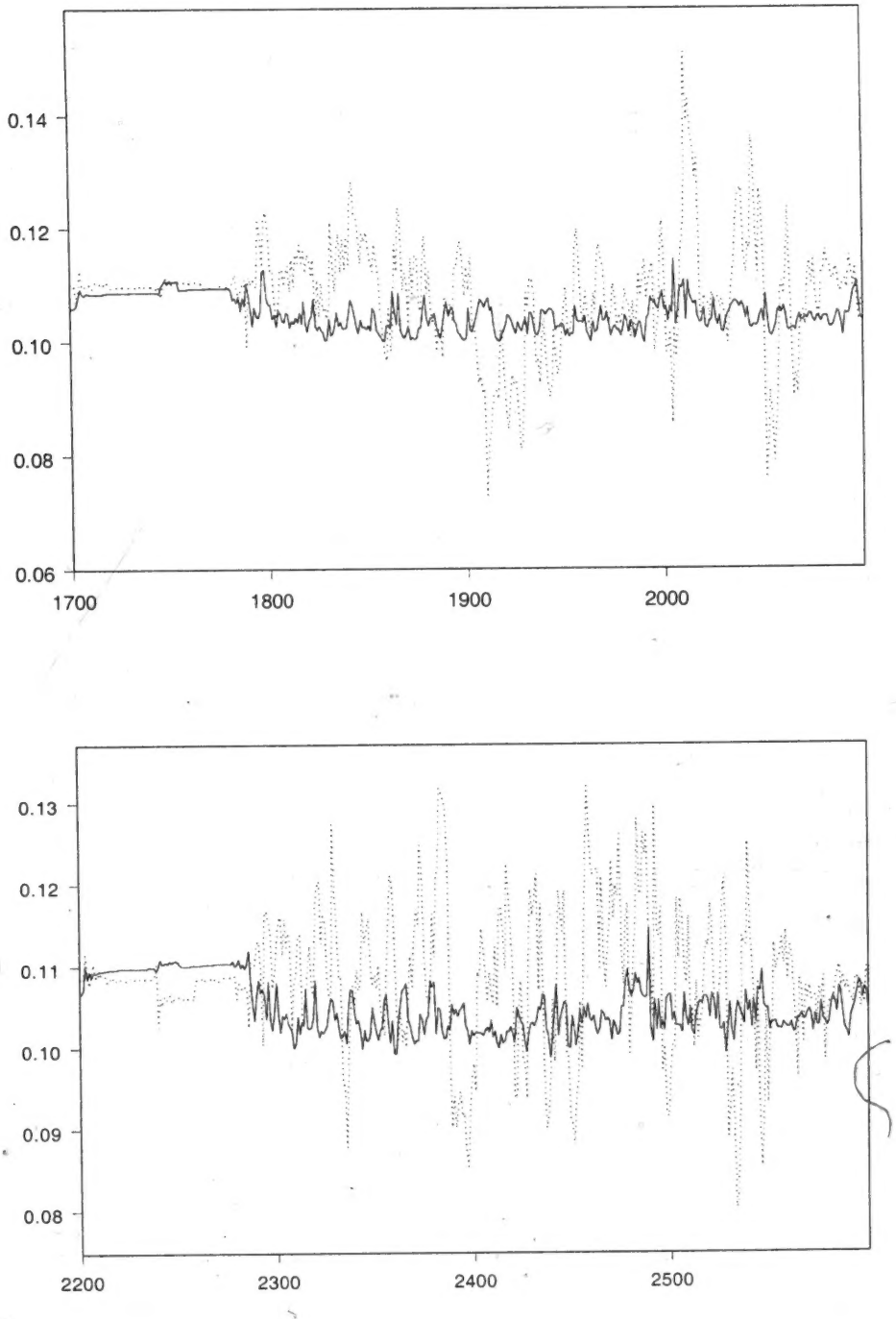


Figure 4.24 TS model, output y3 (floor Z acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data.

4.5.3. Analysis of results

In most cases, the models with 2 clusters perform better than the others within the same set of experiments. Although with higher number of clusters, the FCM process converges to a lower value of the FCM performance index, however, with each additional cluster the working matrix grows by m columns (m is number of outputs), i.e. for this particular dataset by 6×2155 entries. Generally, the bigger is the working matrix **X**, the more unstable becomes the inverse matrix solution $(\mathbf{X}^T\mathbf{X})^{-1}$, which is used for computing the regression parameters as in (4.42). With increasing number of clusters, the algorithm fails more and more on computing the inverse matrix. The inverse matrix was checked through multiplying $(\mathbf{X}^T\mathbf{X})$ by $(\mathbf{X}^T\mathbf{X})^{-1}$. The resulting matrix was supposed to be identity matrix **I**, but the higher was the number of clusters, the more it differed from the identity matrix, with the whole areas consisting of very big numbers instead of ones and zeros. The quality of the inverse matrix used for computing the regression parameters, affects therefore the performance of the resulting model.

As for the FCM termination criteria, with its value of 0.00001, the cluster prototypes were just slightly different from those when FCM stopped after reaching the termination criteria of 0.01. The resulting regression parameters appeared slightly different as well, but taking into account the size of the working matrix **X** to which they were applied, the small difference in parameters caused the substantial difference in the total performance index of the model in both cases.

The fact that in some cases the testing results are similar or even better than training, can be probably explained by the instability of the inverse matrix solution, and also partly by the similarity of the working cycles in the training and testing parts of the dataset.

Same as in most previous experiments (neural networks, local regressions), the model performance is acceptable for outputs y1, y2, y4, and y5, while it practically fails on the outputs y3 and y6.

4.6. Conclusions

In the experiments with the models based on data partition into clusters, both for the local linear regression and Takagi – Sugeno model, the results were acceptable mostly with minimum number of clusters, best of all with 2 clusters. Further increase of the number of clusters did not make much improvement, if any at all. The FCM algorithm just created the groups of almost identical prototypes when the data were partitioned into more than 3 clusters. Overall, local linear regression results were better than those of Takagi – Sugeno model, but both of them worse than neural networks.

In fact, the best results of cluster-based models are quite comparable with those obtained for most static neural network models discussed and experimented in Chapter 2, but introducing a temporal feedback into the neural network as described in the section 2.3.8 had crucial influence on improving the model performance. This concept is not applicable to the cluster-based models discussed in this chapter, and none of them were able to approach the level of performance achieved by the neural network model with feedback.

Compared with neural networks, the performance index had higher variance. Changing the number of clusters and other model parameters resulted in much different data clusters and therefore much different inverse matrices, which had big influence on the regression coefficients and therefore on the performance indices.

Chapter 5. Radial basis function neural networks

5.1. Theoretical base of the model

5.1.1. Concept overview

Radial basis function (RBF) networks originated from multidimensional interpolation models and have been described in the literature since the 1980's [16, 17]. They deal with the general approximation problem common to supervised learning neural networks, namely, given a finite set of training pairs (x, y), find a mapping function f that links the n-dimensional input space to the m-dimensional output space. The RBF approach gained popularity because of two basic facts [3]. First, unlike most supervised learning neural networks which usually find only a local optimum, the RBF network is able to find the global optimum. Second, the training time for RBF networks is much shorter than for most other neural networks, most notably when compared to using the back-propagation rule for adjusting the weights. In addition, the topology of the RBF network is very simple to set up and requires no guessing as compared to back-propagation models [3].

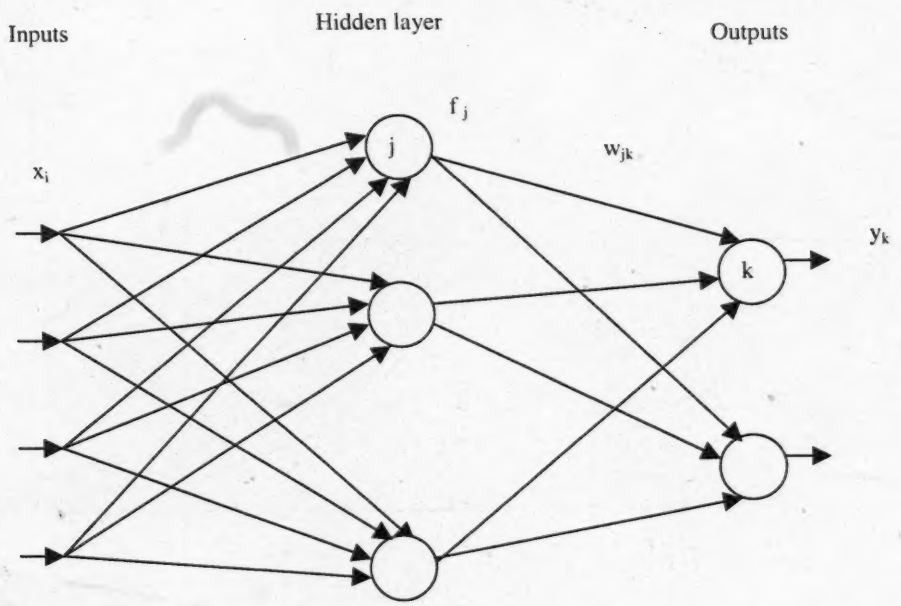
There is a conceptual difference in the classification principle employed by RBF networks compared to hyper-plane classifiers such as multi-layer feed-forward neural networks that use hyper-planes to separate different data points in the dataset from each other. In contrast, the RBF networks use higher order decision functions, namely hyper-spheres, on the same dataset, and are called kernel classifiers [3]. Unlike sigmoidal units covering the entire input space, a single RBF unit covers only a small local region of the input space. This can be an advantage because of faster learning, and also RBF units may form a better set of basis functions to model the input space than sigmoidal units, although it is highly problem-dependent. On the negative side, the locality property of RBF networks may be a disadvantage particularly in high dimensional spaces because many units are needed to cover the entire input space [19].

RBF network can be described as a parameterized model used to approximate an arbitrary function by means of a linear combination of basis functions [6]. RBF networks belong to the class of kernel function networks where the inputs to the model are passed through kernel functions that limit the response of the network to a local region of the input space for each kernel, or basis function. The output from each basis function is weighted and summed and possibly offset by some amount to provide the output of network [6].

While there has been much success in developing nonlinear network models, in particular multi-layer sigmoidal neural networks employing the back-propagation algorithm for training, researchers and practitioners have found that in some cases it can be difficult to train a multi-layer neural network and obtain good performance without resorting to sophisticated algorithms [6]. The main reason is that such networks are nonlinear in the parameters, which means that their learning algorithms need to use the iterative gradient descent approach to solve the nonlinear optimization problem. In contrast, the RBF network models employ algorithms able to overcome the training problem that normally occurs in multi-layer networks.

5.1.2. Topology of the RBF network

The realization of radial basis functions within the framework of neural networks was proposed in [17] and [11]. An RBF network is always composed of three layers – an input layer, a hidden layer and an output layer (Figure 5.1).



x_i (i = 1, 2, …, n) – inputs of the network

f_j (j = 1, 2 ,…, c) – basis functions (elements of the hidden layer)

y_k (k = 1, 2, …, m) – outputs of the network

w_{jk} – weight between unit j of the hidden and unit k of the output layer

Figure 5.1 Topology of the RBF network with n inputs, c basis functions, and m outputs

The input data are directly submitted to each neuron of the hidden layer, without weights. Since the input layer only distributes the model inputs to the hidden layer, it can be stated that an RBF network generally consists of two weight layers – the hidden layer and the output layer [6]. The neurons of the hidden layer compute radial basis function on the input data, and neurons of the output layer perform linear summation of the intermediate outputs of all hidden layer neurons multiplied by corresponding connection weights w_j. The hidden layer differs from other neural networks in that each neuron represents a data cluster that is centered at a particular point with a given radius [3]. Each neuron in the hidden layer calculates the distance between its cluster centre

$\mathbf{v}_j$ and the input vector $\mathbf{x}$, then transforms it through the radial basis function f_j into the neuron output. This output is multiplied by a weight value w_{jk} and fed into the output layer that linearly combines the outputs of the hidden layer and calculates the total output value y_k.

5.1.3. Mathematical description of RBF network operation

The model can be described with the following equation [6]:

$$y_k = w_{0k} + \sum_{j=1}^{c} w_{jk} f(\| \mathbf{x} - \mathbf{v}_j \|) \tag{5.1}$$

where

- f is the radial basis function,
- w_{jk} are the output layer weights,
- w_0 is the output offset,
- $\mathbf{x}$ represents the vector of model inputs
- $\mathbf{v}_j$ are the centres (prototypes) associated with the basis functions
- c is the number of basis functions in the network
- $\| \cdot \|$ denotes the distance

The basic operation of an RBF network can be thought of as follows [6]. Each input vector $\mathbf{x}$ passed to the network is shifted according to some stored parameters (the centres, or the prototypes) in the network. The Euclidian norm is computed for each of these shifted vectors $\mathbf{x} - \mathbf{v}_j$ ($j = 1, 2, \ldots, c$). Each $\mathbf{v}_j$ is a vector with the same number of elements as the input vector $\mathbf{x}$. One prototype is defined for each radial basis function in the network. Prototypes closest to the input data vector $\mathbf{x}$ will have the smallest Euclidian distance, with the extreme case of zero if a prototype $\mathbf{v}_j$ exactly coincides with the input vector $\mathbf{x}$. Conversely, when the data are further away from a prototype $\mathbf{v}_j$, the Euclidian distance becomes larger.

The radial basis function of the hidden layer needs to be chosen in such a way that it would transform big Euclidian distances into small values of the output layer, and vice versa, the smaller Euclidian distances into bigger output values. Thus, for the data points situated far away from the prototypes, the output from the corresponding basis functions will be small, approaching zero with increased distance. Conversely, for the data points close to the prototypes, the output from the corresponding basis functions will be larger, approaching a certain limit value (usually 1, like in a case of Gaussian function) with decreased distance.

The choice of basis function may be dictated by particular implementation requirements or constraints, although from a theoretical perspective, the choice is arbitrary, provided it is restricted to functions that have characteristic "bump" shape in order to be used in this particular architecture [6]. However, the most common choice for basis function is the Gaussian (Figure 5.2):

$$f(x) = e^{\frac{-\|x-v\|^2}{\sigma^2}} \tag{5.2}$$

where

v is the centre (prototype) associated with the basis functions

σ is the radius (standard deviation) of the basis function

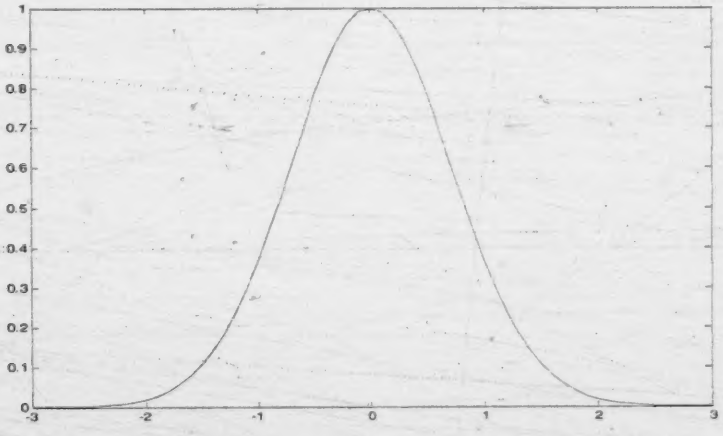


Figure 5.2 The Gaussian radial basis function with centre v = 0 and radius σ = 1

There are several other functions with similar properties, besides Gaussian, described in the literature [3, 6], which can be used as radial basis functions. Some of them are the following:

Multiquadric:

$$f(x) = \sqrt{\| \mathbf{x} - \mathbf{v} \|^2 + \sigma^2} \tag{5.3}$$

Inverse multiquadric:

$$f(x) = \frac{1}{\sigma \cdot \sqrt{\| \mathbf{x} - \mathbf{v} \|^2 + \sigma^2}} \tag{5.4}$$

Cauchy function:

$$f(x) = \frac{1}{\sigma \cdot \left(\| \mathbf{x} - \mathbf{v} \|^2 + \sigma^2 \right)} \tag{5.5}$$

Thin plate spline:

$$f(x) = \| \mathbf{x} - \mathbf{v} \|^2 \cdot \ln \| \mathbf{x} - \mathbf{v} \| \tag{5.6}$$

Cubic spline:

$$f(x) = \| \mathbf{x} - \mathbf{v} \|^3 \tag{5.7}$$

Linear spline:

$$f(x) = \| \mathbf{x} - \mathbf{v} \| \tag{5.8}$$

5.1.4. Training the RBF network

Before training the RBF network, all values in each input in the original dataset must be normalized into [0,1] interval, so that none of the inputs could dominate over the others in the process of computing the distances. Then, several parameters of the RBF network must be determined, such as:

1. Number of basis functions (hidden layer neurons)
2. Centres (prototypes) corresponding to each hidden layer neuron
3. Radii of the hidden layer neurons (standard deviation of the radial basis functions)

As for determining the number of basis functions (or, hidden layer neurons), some formal criteria minimizing a certain objective function were proposed in literature, e.g. [9]. Although, in general, there are no effective methods with proven results for determining the number of clusters for data partition, especially for large datasets with multidimensional input space. However, some data visualization endeavours might help the user to understand, or at least, give some insight into how the data are grouped and decide on the number of clusters that would be logical for this particular dataset. Statistical methods such as Principal Component Analysis (PCA) applied to the analyzed data might also provide some clues for the number of clusters that would appear more or less reasonable for performing the data partition.

To obtain the RBF prototypes, clustering algorithms such as the FCM (Fuzzy-C-Means) [12] can be applied to the training part of the dataset. In result, the data become grouped into (c) clusters and their prototypes are determined as the Euclidian centres of each cluster. Also, the FCM algorithm produces a fuzzy partition matrix describing fuzzy membership of each data point in each cluster on a scale between 0 and 1.

The radii can be determined by the P-nearest neighbours method (4.27) [3].

The output of each neuron of the hidden layer is computed as a Gaussian function (5.2), or one of the other radial basis functions (5.3) – (5.8), defined by this particular cluster prototype and its radius.

After obtaining the values of the radial basis functions, all outputs of the hidden layer neurons need to be weighted and summed to obtain each of the outputs of the neural network. For each data point, the difference between target and real output needs to be computed to evaluate the model performance. The performance index such as RMSE – Root Mean Squared Error (3.3) can

serve as an averaged value of all these differences, throughout all m outputs and all N data points of training dataset.

The connection weights could be then found through iterative training from the randomly initiated weights. In this method, all the connection weights between each neuron of the hidden layer and each neuron of the output layer are first initialized with random values in the interval [0, 1]. The outputs of the neural network are obtained for each data point of the training dataset, to compare them with the target outputs.

The random weights are adjusted in the iterative process of training the neural network. In each iteration, each weight between unit j of the hidden layer and unit k of the output layers is adjusted by a value equal to the output of this particular hidden unit (radial basis function $f_j(x)$) multiplied by the absolute error Δ_k, and also multiplied by learning rate α:

$$w_{jk}(\text{iter}+1) = w_{jk}(\text{iter}) + \alpha \cdot f_j(x) \cdot \Delta_k \tag{5.9}$$

Then, at the end of each iteration, the outputs of the neural network are computed with the new connections, and a new value of the performance index is computed for these new outputs, to check for the termination criterion.

Finally, the model needs to be tested on the testing part of the dataset. The connection weights computed as a result of training indicate how strong the influence of each of the clusters is on each of the outputs. With connections existing after the training, the outputs of the neural network are computed as in (5.1) for each data point of the testing dataset and compared with target output to compute performance index, such as RMSE (3.3) on the testing dataset.

5.1.5. Computing the weights through least square solution

Since the hidden layer outputs are linearly summed in the output layer of the network, there exists a unique least square solution for the connection weights, which means they can be defined without training.

According to (5.1), for each output y_k (k = 1, 2, …, m):

$$y_k = \sum_{j=1}^{c} w_{jk} f_j(x) = w_{1k} f_1(x) + w_{2k} f_2(x) + \ldots + w_{ck} f_c(x) \qquad (5.10)$$

If the model is fed with N data samples $\mathbf{x}_i$, each producing a vector of radial basis functions $\mathbf{f}_i$, and an output value y_{ik} for the output k (i = 1, 2, …, N; and k = 1, 2, …, m), then in matrix notation, (5.10) can be written as follows:

$$\mathbf{y}_k = \mathbf{F}\mathbf{w}_k \qquad (5.11)$$

where

$\mathbf{y}_k$ is an N-dimensional vector of values predicted by the model for the particular output k (k = 1, 2, …, m):

$$\mathbf{y}_k = \begin{bmatrix} y_{1k} \\ y_{2k} \\ \ldots \\ y_{Nk} \end{bmatrix} \qquad (5.12)$$

F is an N×c matrix of RBF values (outputs of the hidden layer):

$$\mathbf{F} = \begin{bmatrix} f_{11} & f_{12} & \ldots & f_{1c} \\ f_{21} & f_{22} & \ldots & f_{2c} \\ \ldots & \ldots & \ldots & \ldots \\ f_{N1} & f_{N2} & \ldots & f_{Nc} \end{bmatrix} \qquad (5.13)$$

$\mathbf{w}_k$ is a c-dimensional vector of connection weights to be defined for the particular output k (k = 1, 2, …, m):

$$\mathbf{w}_k = \begin{bmatrix} w_{1k} \\ w_{2k} \\ \dots \\ w_{ck} \end{bmatrix} \tag{5.14}$$

If $\mathbf{t}_k$ is a vector of target values for output k already existing in the dataset:

$$\mathbf{t}_k = \begin{bmatrix} target_{1k} \\ target_{2k} \\ \dots \\ target_{Nk} \end{bmatrix} \tag{5.15}$$

then, according to (4.14), vector of errors $\mathbf{e}_k$ is expressed as follows:

$$\mathbf{e}_k = \mathbf{t}_k - \mathbf{y}_k = \mathbf{t}_k - \mathbf{F}\mathbf{w}_k \tag{5.16}$$

To define $\mathbf{w}_k$, we need to minimize the objeetive function as in (4.15) – (4.16):

$$L = \mathbf{e}_k^T \mathbf{e}_k = (\mathbf{t}_k - \mathbf{F}\mathbf{w}_k)^T (\mathbf{t}_k - \mathbf{F}\mathbf{w}_k) =$$

$$= \mathbf{t}_k^T \mathbf{t}_k - \mathbf{w}_k^T \mathbf{F}^T \mathbf{t}_k - \mathbf{t}_k^T \mathbf{F}\mathbf{w}_k + \mathbf{F}^T \mathbf{F}\mathbf{w}_k^T \mathbf{w}_k =$$

$$= \mathbf{t}_k^T \mathbf{t}_k - 2\mathbf{w}_k (\mathbf{F}^T \mathbf{t}_k) + \mathbf{w}_k^T (\mathbf{F}^T \mathbf{F})\, \mathbf{w}_k \tag{5.17}$$

which means to differentiate it with respect to each element of $\mathbf{w}_k$, then set each derivative equal to 0 and solve each equation as in (4.17) – (4.19):

$$\nabla_{w_k} L = -2\mathbf{F}^T \mathbf{t}_k + 2\mathbf{F}^T \mathbf{F}\mathbf{w}_k = 0 \tag{5.18}$$

With algebraic operations on (5.18), we obtain the expression for $\mathbf{w}_k$, which represents the least square solution for (5.18), i.e. all the connections w_{1k}, w_{2k}, …, w_{ck}, necessary for predicting the model outputs y_k as in (5.10):

$$\mathbf{w}_k = (\mathbf{F}^T \mathbf{F})^{-1} \mathbf{F}^T \mathbf{t}_k \tag{5.19}$$

The formula (5.19) is used to find the connections for each of the m outputs separately. After defining all the connections, they are used to for computing the model outputs for the testing dataset as in (5.10) and then performance index such as RMSE (3.3) on the testing dataset.

5.2. Applying the model to the analyzed dataset

5.2.1. Experimenting with the number of clusters and FCM termination criteria

From all the data visualization techniques described above, it did not become clear what number of clusters would be optimal for this dataset. In this research, number of clusters (i.e. number of hidden layer neurons) was varied from 2 to 24. Table 5.1 presents the performance of the model with changing number of clusters on the training and testing datasets. Three values of the FCM termination criterion were tried: 0.01, 0.001, and 0.0001. The FCM termination criterion means that the FCM algorithm as an iterative process of updating the cluster prototypes and the partition matrix stops after the index Q (5.2) computed in the current iteration differs from the one computed in the previous iteration less than by the termination criterion value.

Table 5.1 Performance of RBF networks with different number of clusters

FCM stop	0.01		0.001		0.00001	
# of clust.	Training	Testing	Training	Testing	Training	Testing
2	0.018527	0.019298	0.018487	0.019268	0.018474	0.019258
3	0.015023	0.016060	0.015065	0.016083	0.015088	0.016096
4	0.014293	0.014563	0.014244	0.014518	0.014223	0.014496
5	0.013165	0.013670	0.013237	0.013741	0.014502	0.014661
6	0.012678	0.012689	0.013006	0.013259	0.013039	0.013529
7	0.012731	0.012630	0.012736	0.012660	0.012889	0.012894
8	0.012596	0.012571	0.012823	0.012762	0.013051	0.012953
9	0.011762	0.012685	0.011490	0.012487	0.012971	0.012975
10	0.011855	0.011252	0.011934	0.011336	0.012246	0.012661
11	0.011697	0.012381	0.012952	0.013115	0.010054	0.010953
12	0.009621	0.010650	0.010453	0.011539	0.012331	0.013028
13	0.008823	0.009658	0.008745	0.009589	0.008588	0.009374
14	0.009197	0.009661	0.008803	0.009291	0.009288	0.009679
15	0.008401	0.008870	0.008458	0.009061	0.008293	0.008978
16	0.007527	0.008454	0.007465	0.008269	0.007197	0.008008
17	0.007012	0.007323	0.006997	0.007333	0.007504	0.007183
18	0.007361	0.008883	0.007305	0.008218	0.007291	0.008184
19	0.007201	0.008230	0.007197	0.008014	0.007242	0.008026
20	0.007384	0.008093	0.007341	0.008170	0.007531	0.007925
21	0.007393	0.007944	No solution	No solution	No solution	No solution
22	0.007493	0.008082	0.007432	0.008002	0.007800	0.008247
23	0.006373	0.007244	0.006367	**0.007149**	0.006495	0.007185
24	0.007026	0.007555	0.007055	0.008277	No solution	No solution

With the FCM termination criterion set to 0.01, the algorithm converged after 20 to 25 iterations. For this value, model training and testing performance indices are plotted on the Figure 5.3. The best performance in this series of experiments was reached with 23 clusters.

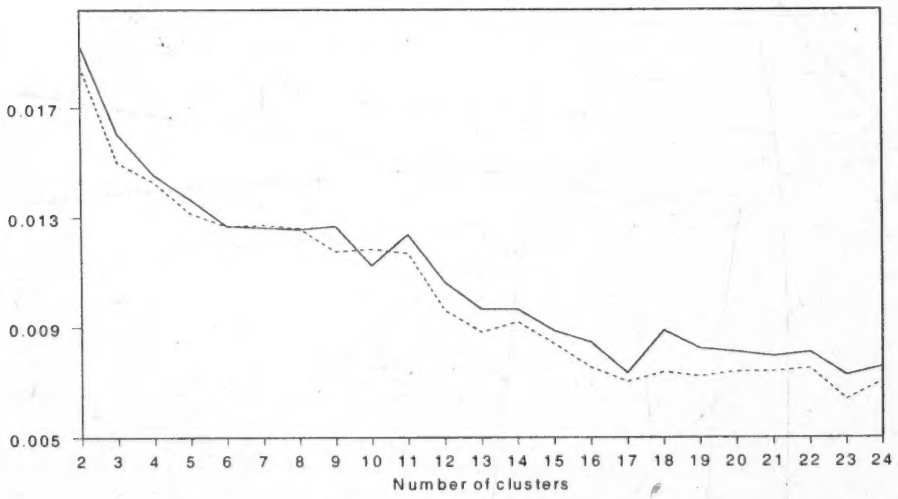


Figure 5.3 RBF network performance index RMSE as a function of number of clusters in data partition: dotted line – training RMSE, solid line – testing RMSE. Type of RBF – Gaussian.

To perform the cross-validation (repeating the experiments for 10 different data partitions to obtain average performance), besides considering the original working dataset as a time series, the data partition into training and testing parts was performed randomly 9 more times. Each time, all the data were randomly shuffled by consecutive blocks of 10 data points, and the first 2155 data points were picked for the training dataset. The remaining 1015 data points created the testing dataset, and were fed into the trained network by blocks in random order, to check the ability of the model to react to unpredictable data patterns. The performance indices were averaged throughout the series of 10 experiments (Table 5.2), and the average performance index was plotted on Figure 5.4. The average curve is smoother than that on Figure 5.3, showing more clearly the trend of declining both the training and testing errors with increase of the number of clusters. Apparently, the best average result was achieved with the highest number of clusters that was tried.

Table 5.2 Performance of RBF network for the original dataset and average performance for 10 different partitions into training and testing parts (original sequence and 9 randomly shuffled).

# of clusters	Original data (training)	Original data (testing)	Average (training)	St. dev. (training)	Average (testing)	St. dev. (testing)
2	0.018527	0.019298	0.019332	0.000986	0.019901	0.001115
3	0.015023	0.016060	0.015390	0.000678	0.015967	0.001045
4	0.014293	0.014563	0.014558	0.000548	0.015168	0.000944
5	0.013165	0.013670	0.014199	0.000697	0.014825	0.000826
6	0.012678	0.012689	0.013153	0.000806	0.013821	0.000942
7	0.012731	0.012630	0.012954	0.000437	0.013782	0.000825
8	0.012596	0.012571	0.012502	0.001037	0.013202	0.001181
9	0.011762	0.012685	0.011085	0.000850	0.011961	0.001373
10	0.011855	0.011252	0.010256	0.001123	0.011201	0.001448
11	0.011697	0.012381	0.009341	0.000578	0.010311	0.001090
12	0.009621	0.010650	0.009155	0.000483	0.010004	0.000526
13	0.008823	0.009658	0.008922	0.000554	0.009896	0.000600
14	0.009197	0.009661	0.008338	0.000452	0.009170	0.000330
15	0.008401	0.008870	0.008269	0.000633	0.009129	0.000736
16	0.007527	0.008454	0.008012	0.000388	0.008826	0.000497
17	0.007012	0.007323	0.007470	0.000456	0.008255	0.000364
18	0.007361	0.008883	0.007518	0.000314	0.008504	0.000387
19	0.007201	0.008230	0.007660	0.000337	0.008469	0.000383
20	0.007384	0.008093	0.007225	0.000303	0.008144	0.000380
21	0.007393	0.007944	0.007332	0.000374	0.008097	0.000302
22	0.007493	0.008082	0.007078	0.000223	0.007864	0.000239
23	0.006373	0.007244	0.007129	0.000303	0.007960	0.000432
24	0.007026	0.007555	0.006924	0.000185	0.007856	0.000138

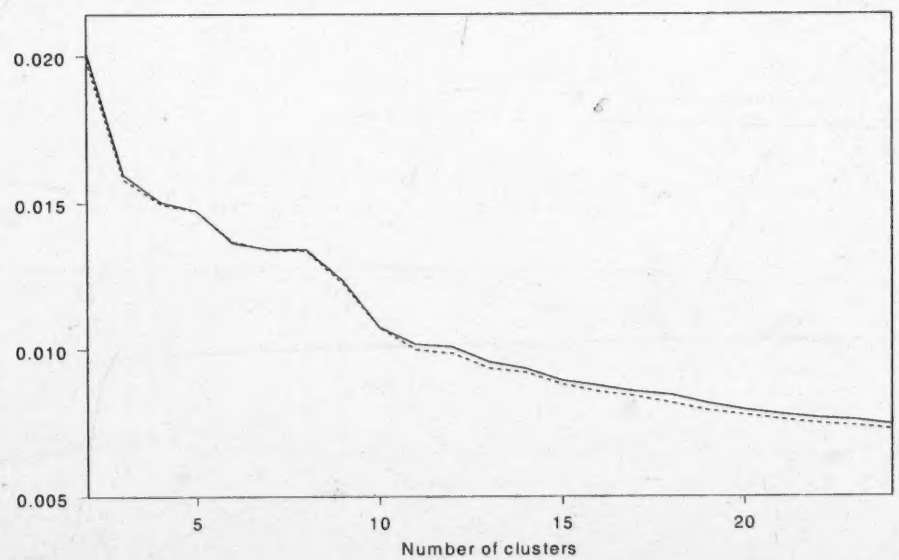


Figure 5.4 Average (for 10 data partitions) performance index RMSE as a function of number of clusters: dotted line – training RMSE, solid line – testing RMSE. Type of RBF – Gaussian.

Here, the weights were computed through least square solution. Clustering was performed only for the inputs. The experiments with both the inputs and the outputs taken together into consideration in the clustering process resulted in the absence of inverse matrix solution for the most configurations, especially with higher number of clusters, while not bringing any performance improvements for the configurations where the solution existed.

In this series of experiments, radial basis functions are taken as Gaussian. The value of Gaussian function is computed according to (5.2) for each datapoint $\mathbf{x}_k$ based on its Euclidian distance to this particular prototype $\mathbf{v}_j$.

$$f_{kj}(x) = e^{\frac{-\|x_k - v_j\|^2}{\sigma_j^2}} \tag{5.20}$$

5.2.2. Experimenting with types of radial basis functions

Besides Gaussian, the models with the other types of radial basis functions (5.3) – (5.8) were run through the same experiments. The results are summarized in the

Table 5.3, and for one of the models, with inverse multiquadric function, the model training and testing performance indices are plotted on the Figure 5.5. The experiments confirmed the trend of improving the results with increasing the number of clusters. However, for the most functions, the improvement is most noticeable before the number of clusters reaches 10 – 15. Beyond that, the performance of the model practically stabilizes at some achieved level and it makes little sense to increase the number of clusters further, to obtain some negligible gain in performance.

Table 5.3 Performance of models with different types of radial basis functions

RBF type	Multiquad.		Inv. multiq.		Cauchy	
# of clust.	Training	Testing	Training	Testing	Training	Testing
2	0.008405	0.008870	0.008065	0.00847	0.012477	0.012692
3	0.006813	0.007241	0.007189	0.007664	0.010336	0.010943
4	0.006639	0.006944	0.006958	0.00736	0.009833	0.010148
5	0.005985	0.006315	0.006645	0.007134	0.009187	0.009719
6	0.005841	0.006164	0.006512	0.006926	0.008850	0.009176
7	0.005730	0.006094	0.006468	0.006816	0.008866	0.009052
8	0.005628	0.006046	0.006421	0.006769	0.008814	0.009002
9	0.005593	0.005956	0.006177	0.006836	0.008354	0.009111
10	0.005524	0.006054	0.006258	0.006583	0.008410	0.008442
11	0.005466	0.005938	0.006177	0.006735	0.008571	0.009072
12	0.005427	0.005872	0.005773	0.006379	0.007373	0.008093
13	0.005372	0.005893	0.005694	0.006251	0.006995	0.007621
14	0.005372	**0.005884**	0.005707	0.006215	0.007066	0.007551
15	0.005332	0.005920	0.005662	0.006176	0.006817	0.007316
16	0.005338	0.005918	0.005578	0.006190	0.006546	0.007207
17	0.005315	0.005913	0.005516	0.006025	0.006303	0.006672
18	0.005325	0.005892	0.005546	0.006188	0.006512	0.007193
19	0.005321	0.005963	0.005533	0.006159	0.006384	0.006899
20	0.005310	0.005981	0.005507	0.006134	0.006377	0.006985
21	No solution	No solution	0.005501	0.006118	0.006382	0.006953
22	0.005295	0.005960	0.005511	0.006066	0.006482	0.006984
23	0.005287	0.005888	0.005428	0.006059	0.006089	0.006688
24	No solution	No solution	0.005450	0.006077	0.006257	0.006782

RBF type	Thin plate		Cubic spl		Linear spl	
# of clust.	Training	Testing	Training	Testing	Training	Testing
2	0.030099	0.027099	0.030906	0.031634	0.013156	0.013131
3	0.018061	0.020794	0.016269	0.019661	0.010797	0.011136
4	0.016764	0.018366	0.016633	0.019383	0.010289	0.010523
5	0.015307	0.018040	0.015088	0.016430	0.009224	0.009759
6	0.015411	0.017276	0.015178	0.015706	0.008768	0.009205
7	0.013454	0.015148	0.012231	0.013702	0.008642	0.009059
8	0.012845	0.013988	0.011559	0.013682	0.008512	0.008951
9	0.013221	0.014640	0.012219	0.013856	0.008493	0.008894
10	0.012374	0.013798	0.011304	0.012331	0.008105	0.008508
11	0.013598	0.015473	0.011359	0.012067	0.008427	0.008851
12	0.012279	0.012890	0.008401	0.008609	0.007575	0.008021
13	0.012147	0.012590	0.008270	0.007881	0.007171	0.007820
14	0.010440	0.011825	0.007305	0.007658	0.006833	0.007536
15	0.011428	0.012046	0.007786	0.007689	0.006649	0.007328
16	0.010994	0.011395	0.007132	0.007517	0.006769	0.007441
17	0.010947	0.011577	0.007370	0.007369	0.006573	0.007332
18	0.010419	0.011189	0.007370	0.007660	0.007191	0.007809
19	0.010223	0.010945	0.006525	0.007260	0.006563	0.007380
20	0.009963	0.011390	0.006782	0.007084	0.006504	0.007348
21	0.009401	0.010860	0.006663	0.006899	0.006453	0.007199
22	0.009374	0.010421	0.006446	0.007125	0.006531	0.007346
23	0.009293	0.011206	0.006362	0.006523	0.006424	0.007300
24	0.009290	0.010533	No solution	No solution	0.006419	0.007036

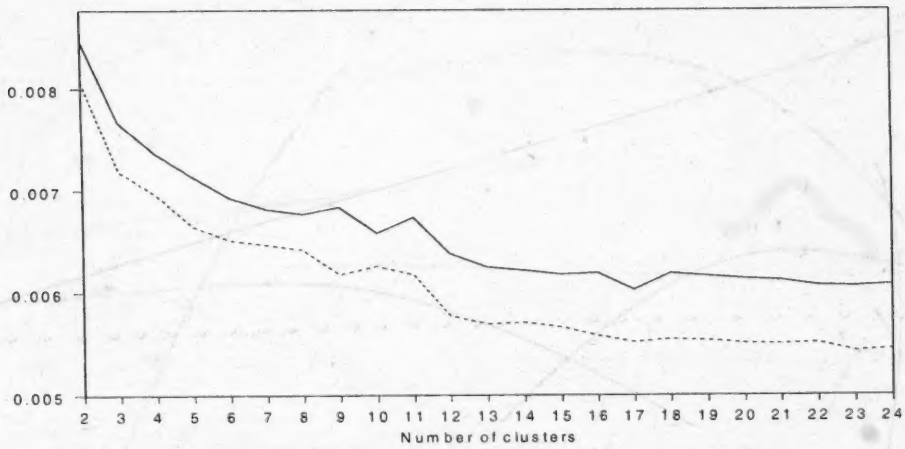


Figure 5.5 RBF network performance index RMSE as a function of number of clusters: dotted line – training RMSE, solid line – testing RMSE. Type of RBF – inverse multiquadric.

5.2.3. Computing RBF as partition matrix values

Finally, instead of computing radial basis functions as Gaussian (5.2) or one of the other functions above (5.3) – (5.8), their values can be also represented by the elements of the partition matrix for each data point obtained as a result of the FCM clustering algorithm. The results of experiments, including ten-fold cross-validation, are summarized in the Table 5.4. In this case, the model performance index practically does not depend on the number of clusters (Figure 5.6), while in the case of Gaussian and other RBF, the performance index improves when increasing the number of clusters (Figure 5.3 and Figure 5.5). With further increase of the number of clusters, the least square solution for the weight connections fails to be found.

As it can be seen from the Table 5.4, cross-validation for 10 random splits of the data into training and testing parts brought poor results. For the shuffled data, model performance index was in average an order of magnitude higher than for the original dataset, and provided no basis for meaningful conclusions.

Table 5.4 Performance of RBF network for the original dataset and average performance for 10 different partitions into training and testing parts (original sequence and 9 randomly shuffled).

# of clusters	Original data (training)	Original data (testing)	Average (training)	St. dev. (training)	Average (testing)	St. dev. (testing)
2	0.005763	0.006588	0.005806	0.000005	0.006637	0.000018
3	0.005742	0.006564	0.005800	0.000006	0.006640	0.000016
4	0.005699	0.006507	0.005794	0.000009	0.006649	0.000019
5	0.005697	0.006506	0.005790	0.000008	0.006652	0.000025
6	0.005686	0.006495	0.005782	0.000011	0.006654	0.000029
7	0.005680	0.006491	0.005778	0.000008	0.006662	0.000030
8	0.005677	0.006478	0.005773	0.000008	0.006668	0.000035
9	0.005671	0.006477	0.005771	0.000009	0.006665	0.000027
10	0.005668	0.006483	0.005764	0.000010	0.006669	0.000033
11	0.005666	0.006477	0.005762	0.000010	0.006671	0.000034
12	0.005668	0.006482	0.005757	0.000016	0.006677	0.000037
13	0.005659	0.006474	0.005752	0.000017	0.006682	0.000039
14	0.005648	0.006492	0.005750	0.000019	0.006680	0.000039
15	0.005655	0.006488	0.005743	0.000013	0.006684	0.000033
16	0.005647	0.006494	0.005746	0.000011	0.006680	0.000045
17	0.005650	0.006467	0.005738	0.000013	0.006697	0.000042
18	0.005643	0.006498	0.005737	0.000014	0.006681	0.000039
19	0.005638	0.006477	0.005732	0.000017	0.006687	0.000045
20	0.005630	0.006486	0.005721	0.000026	0.006699	0.000041
21	0.005625	0.006483	0.005725	0.000018	0.006708	0.000053
22	0.005624	0.006457	0.005720	0.000018	0.006700	0.000038
23	0.005763	0.006469	0.005720	0.000020	0.006708	0.000043
24	0.005742	0.006484	0.005718	0.000016	0.006703	0.000056

The values of radial basis functions taken as the elements of the partition matrix.

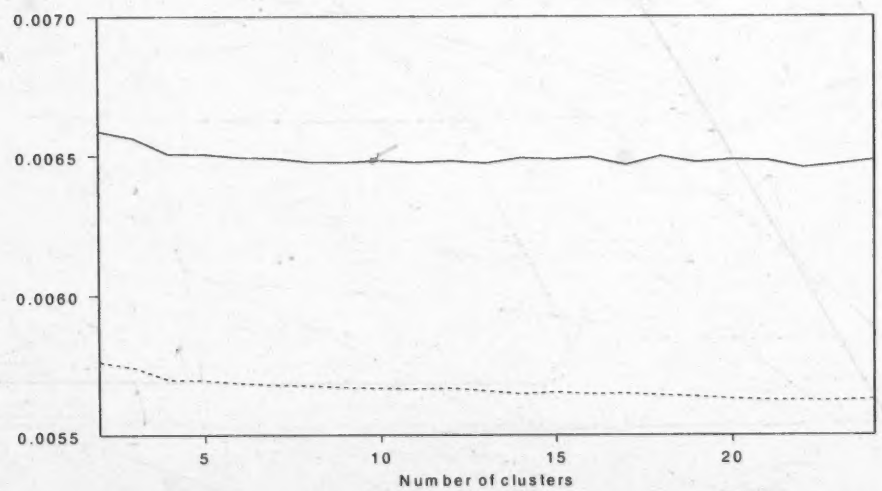


Figure 5.6 RBF network performance index RMSE as a function of number of clusters: dotted line – training RMSE, solid line – testing RMSE. Values of RBF are taken from partition matrix.

5.2.4. Experimenting separately on different outputs

RBF networks, same as feed-forward neural networks and linear models, also delivered different performance on the outputs y1, y2, y4, and y5 from one side (acceleration dimensions X and Y), and on the outputs y3 and y6 from the other side (acceleration dimension Z), when separated from each other. First, the experiments were conducted on the dataset containing all 8 inputs and only the outputs y1, y2, y4, and y5. The performance results were as follows:

# of clusters	4 clusters	12 clusters	20 clusters
Training RMSE	0.004242	0.004161	0.004108
Testing RMSE	0.004571	0.004502	0.004456

Then the same experiments were repeated on the dataset containing same inputs and only the outputs y3 and y6, with the following results:

# of clusters	4 clusters	12 clusters	20 clusters
Training RMSE	0.007864	0.007857	0.007836
Testing RMSE	0.009262	0.009252	0.009298

Again, in all these experiments, the results on the outputs y3 and y6 were always worse than for the outputs y1, y2, y4, and y5 (RMSE about twice as higher).

5.2.5. Experimenting with dynamic RBF network models

The concept of feedback (as described in 3.2.3) was applied to the working dataset to explore RBF network behaviour as a dynamic model. Target outputs corresponding to each data point were introduced as additional inputs for each next data point fed into the model, thus creating previous cycle feedback. The experiments, including ten-fold cross-validation, demonstrated chaotic

performance of the model, in the most cases resulting in the absence of least square solution for the network connections. Experimenting with two-cycle feedback (extra inputs taken from two previous data points target outputs) provided no meaningful results either.

5.2.6. Computing weights through iterative learning starting from random values

The method of iterative learning, starting from random weight values, results in very similar matrix of weights and practically identical performance index, as when the connections are defined through the least square solution. This method always provided a solution, even where the least-square method failed, however, it required experimenting with the learning rate and termination criterion. For different number of clusters, there were different optimal values of learning rate. Also, it took more execution time, since the training iterative process converged after a few dozen iterations.

Figure 5.7, Figure 5.8, and Figure 5.9 present some of the outputs (y1 – y3) of this model vs. targets for the training and testing data. The plots are built on the fragments of the training and testing datasets, with number of clusters 23, and FCM termination criterion of 0.01. For this RBF network, training RMSE was 0.006848, testing RMSE was 0.007861

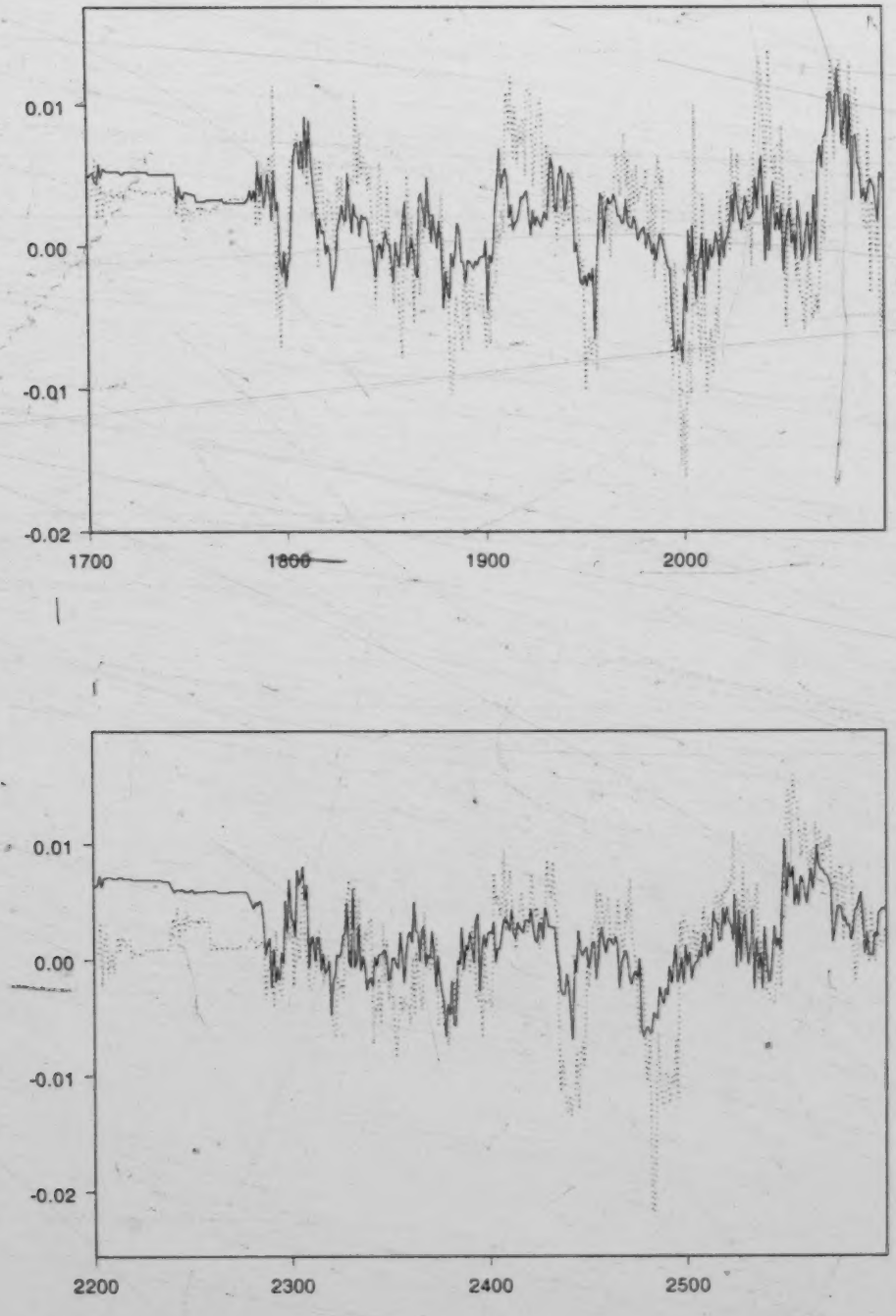


Figure 5.7 RBF model, output y1 (floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data.

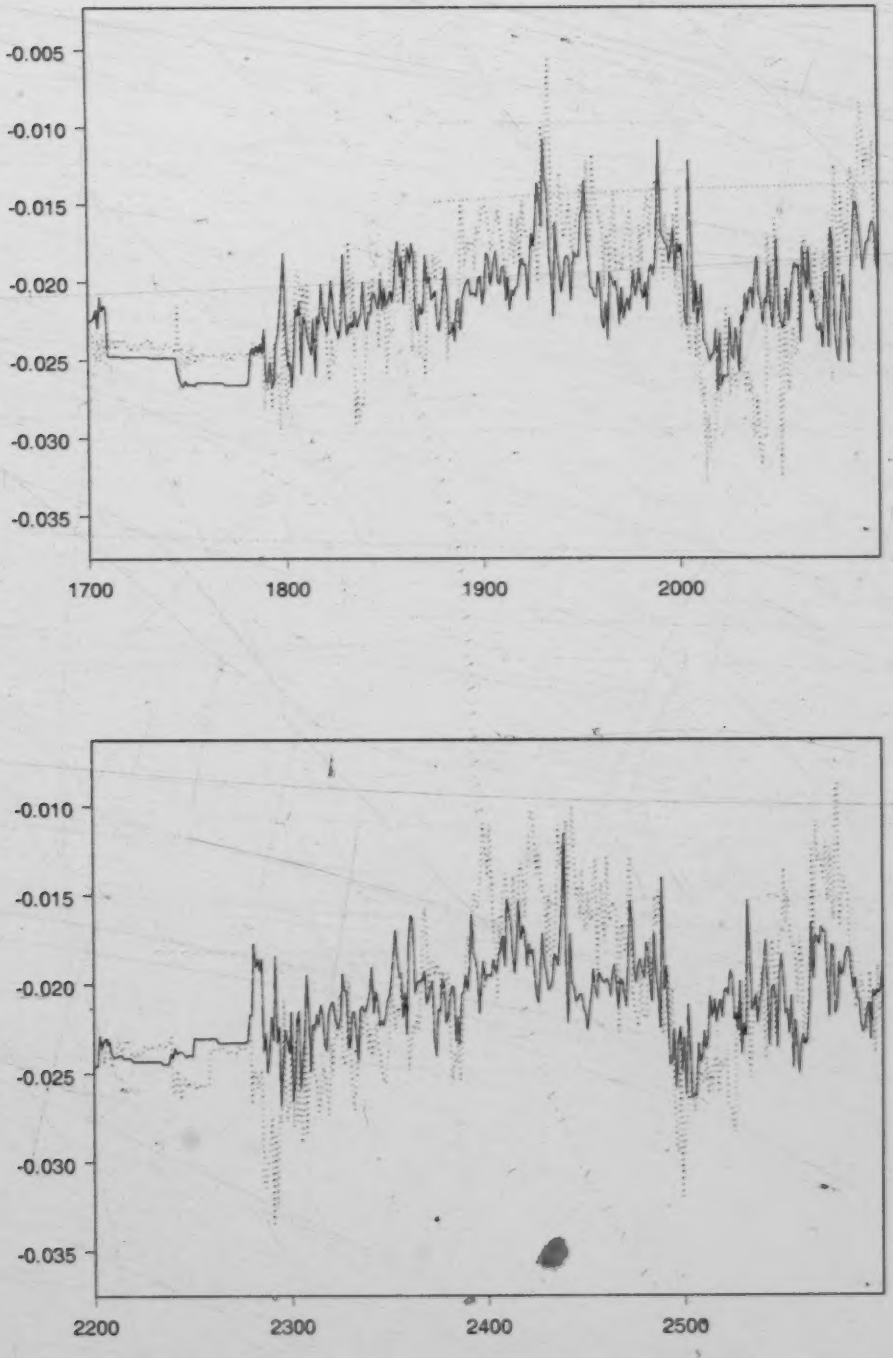


Figure 5.8 RBF model, output y2 (floor Y acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data.

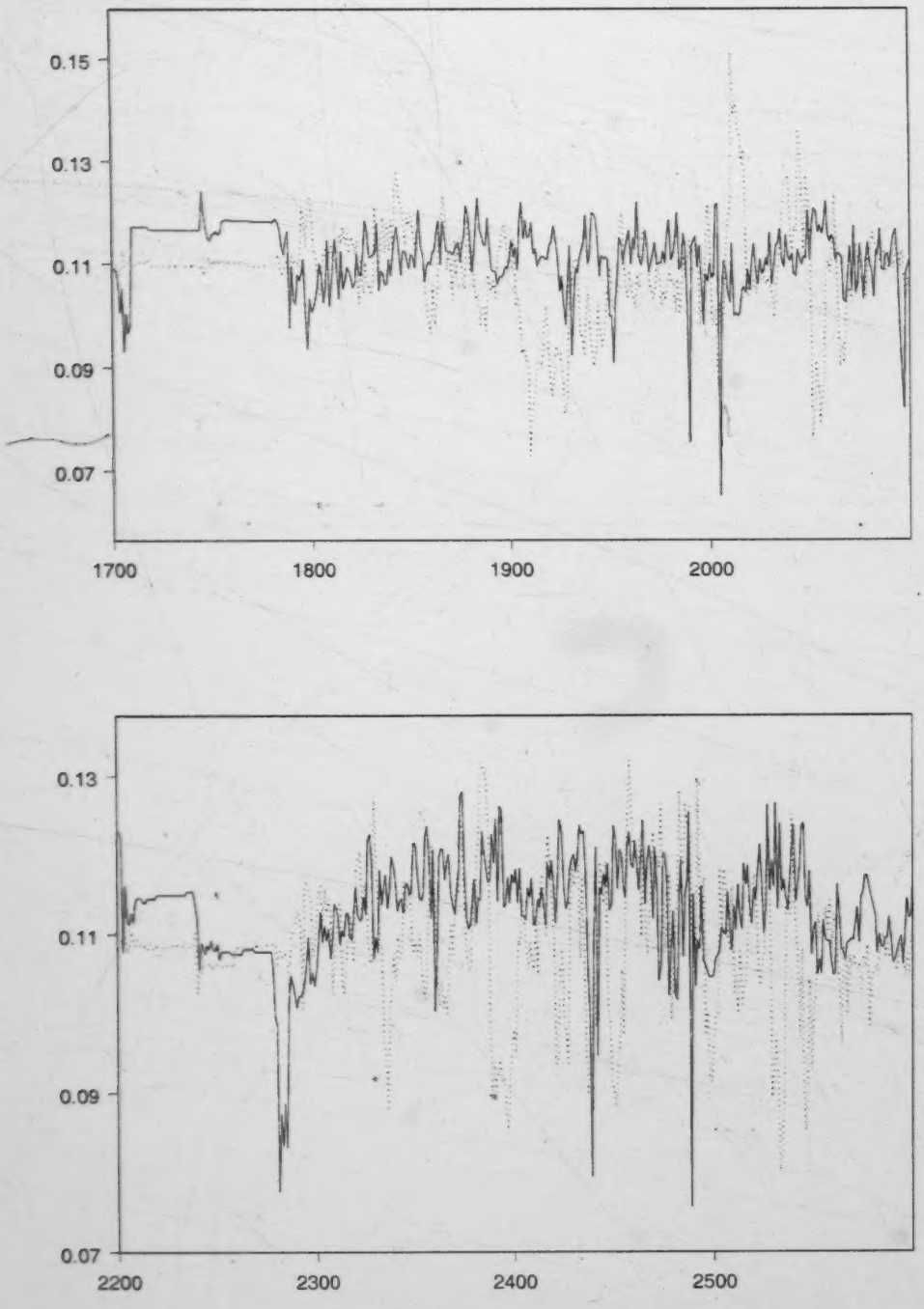


Figure 5.9 RBF model, output y3 (floor X acceleration): dotted line – target, solid line – model output. Upper plot represents a fragment of training data, lower plot – fragment of testing data.

5.3. Conclusions

Experimenting with FCM termination criteria presumably might lead to more precise placing of the prototypes and data allocation into cluster, but the Table 5.1 proved that it did not bring any visible improvement in model performance.

The experiments with the number of clusters demonstrated a clear trend of improving the model performance when the number of clusters increased, which was confirmed through the cross-validation experiments with different data splits for training and testing. The number of clusters was varied up to 24, and further increases could probably bring further improvements to the model performance. In extreme case, each data point could have its own receptive field, to minimize the error. However, having dozens of local models for one dataset contradicts to the idea of getting the acceptable results with the simplest possible architecture and minimum number of receptive fields. So, the number of clusters in the experiments was limited to 24. Naturally, the best average performance in cross-validation experiments was obtained with 24 clusters (testing RMSE = 0.007437, highlighted with boldface in the Table 5.2).

These were the results achieved by the model with Gaussian radial basis function. The experiments with different types of radial basis function showed that much better performance was obtained for the model with multi-quadric radial basis function compared to the others. The model with 14 clusters obtained testing RMSE = 0.005884, highlighted with boldface in the Table 5.3. This is comparable to the best results achieved by the other types of models considered in previous chapters (except the neural network with feedback).

In the experiments where the values of the elements of the partition matrix were used instead of computing radial basis functions, the performance was somewhere between the levels achieved by the models with Gaussian and multi-quadric radial basis functions, and it did not change when increasing the number of clusters (Figure 5.6).

Chapter 6. General conclusion

Comparative analysis of all types of the models discussed in this research, with the best results achieved by each type in terms of performance index on the testing dataset are presented in the Table 6.1 and Figure 6.1.

- Table 6.1 Comparative analysis of performance of all types of the models

#	Type and parameters of the model	Q train	St.dev.	Q test	St.dev.
1	Neural network with one hidden layer, 5 neurons, learning rate 0.015, no momentum, static, sigmoid transfer function in hidden and output layer	0.005411		0.005680	
2	For that model, best average result from 10-fold cross-validation (one hidden layer, 11 neurons, learning rate 0.03)	0.006051	0.000065	0.006041	0.000118
3	Neural network with one hidden layer, 3 neurons, learning rate 0.005, no momentum, static, sigmoid transfer function in hidden layer, linear in output layer	0.005472		0.005704	
4	Neural network with one hidden layer, 7 neurons, learning rate 0.005, no momentum, static, arctangent function in hidden layer, linear in output layer	0.005501		0.005796	
5	Neural network with one hidden layer, 16 neurons, learning rate 0.015, no momentum, static, hyperbolic tangent in hidden layer, linear in output layer	0.005943		0.006110	
6	Neural network with one hidden layer, 5 neurons, learning rate 0.02, momentum 0.5, static, sigmoid transfer function in hidden and output layer	0.005389		0.005658	
7	Neural network with two hidden layers, 3 neurons in first hidden layer, 5 in second, learning rate 0.015, no momentum, static, sigmoid transfer function in all layers	0.005462		0.005677	
8	Neural network with one hidden layer, 11 neurons, learning rate 0.02, no momentum, one-cycle feedback, sigmoid function in hidden and output layer	0.003802		0.003982	
9	For one-cycle feedback model, best average result from 10-fold cross-validation (one hidden layer, 11 neurons, learning rate 0.005)	0.005550	0.002584	0.005199	0.002765
10	Neural network with one hidden layer, 7 neurons, learning rate 0.01, no momentum, two-cycle feedback, sigmoid function in hidden and output layer	0.003731		0.003950	
11	For two-cycle feedback model, best average result from 10-fold cross-validation (one hidden layer, 10 neurons, learning rate 0.005)	0.005544	0.002589	0.005245	0.002766

12	Linear regression with parameters computed for the training dataset	0.005407		0.005889	
13	Local linear models, 2 clusters created considering both inputs and outputs, no threshold at data allocation	0.005358		0.005820	
	Local linear models, 2 clusters created considering only inputs, no threshold	0.015471		0.018190	
14	Takagi – Sugeno model, 2 clusters created considering only inputs, FCM termination criterion 0.01	0.006333		0.006407	
15	Takagi – Sugeno model, 2 clusters created considering both inputs and outputs, FCM termination criterion 0.01	0.006636		0.007727	
16	Takagi – Sugeno model, 2 clusters created considering both inputs and outputs, FCM termination criterion 0.00001	0.006724		0.007300	
17	RBF network, 23 receptive fields, Gaussian radial basis function, FCM termination criterion 0.001, weights defined through least squares solution	0.006367		0.007149	
18	RBF network, 14 receptive fields, multi-quadric radial basis function, FCM termination criterion 0.01, weights defined through least squares solution	0.005372		0.005884	
19	RBF network, 22 receptive fields, values of partition matrix instead of radial basis function values, FCM termination criterion 0.01, least squares solution for weights	0.005624		0.006457	
20	RBF network, 23 receptive fields, Gaussian radial basis function, FCM termination criterion 0.01, weights trained from random values instead of least squares solution	0.006848		0.007861	

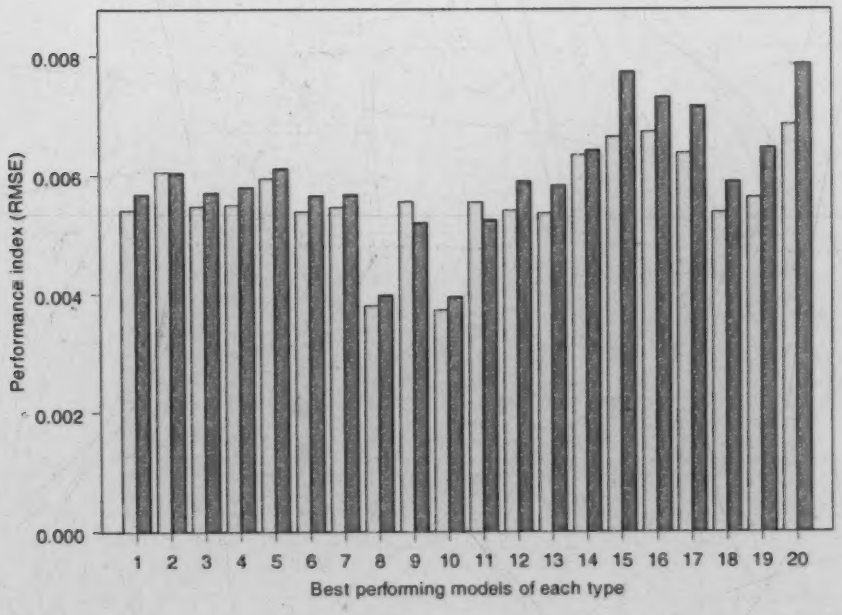


Figure 6.1 Comparison of training and testing performance of all types of the models

Static neural network models achieved only slightly better performance than linear regression. The neural networks with feedback performed significantly better and demonstrated outstanding results compared to all the other types of models. Local linear models, as well as Takagi – Sugeno model, performed acceptably only with the minimum number of clusters. Overall, Takagi – Sugeno model appeared to be the poorest performing type of model among all the others. RBF networks, in contrast, showed clear improvement in performance with increasing the number of receptive fields. Among the RBF network models, the one with multi-quadric radial basis function performed better than the models with Gaussian or other types of functions.

All these conclusions hold just for this particular dataset. The best performing configurations for each type of models were found through the numerous experiments conducted on this dataset, and not necessarily their performance would appear to be the best when applied to the other data. In each particular case, implementing a number of experiments can help to find the type of model that will be able to grasp the nature and structure of the data better than the other types, and then further experiments with the model parameters can lead to the best performing configuration.

The dynamic neural network with 7 neurons in a hidden layer and with 2-cycle feedback from the target outputs represents that particular best performing model. It was defined in result of all the described experiments conducted on the working dataset. The model with feedback demonstrated its superior (compared to all the other models) ability to learn from historically collected data and predict their behaviour. This can be translated into the seat and floor acceleration values predicted for the future working cycles, with the level of precision allowing the company to deal with the issues of operational efficiency and wear of equipment on the basis of knowledge obtained by the model from the available data.

Bibliography

1. G.M.Shepherd, C.Koch. *Introduction to synaptic circuits*, published in *The Synaptic Organization of the Brain*, edited by Gordon M. Shepherd, pp.3-31. New York: Oxford University Press, 1990.

2. S.Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

3. K.Cios, W.Pedrycz, R.Swiniarski. *Data Mining Methods for Knowledge Discovery*. Kluwer Academic Publishers, 1998.

4. S.W.Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. Chapter 26. *Neural Networks (and more!)* California Technical Publishing, 1997.

5. *Electronic Statistics Textbook*, Chapter on *Neural Networks*. Tulsa, OK: StatSoft, Inc., 2001. Retrieved from the Web: http://www.statsoft.com/textbook/stathome.html

6. A.D.Back, Windale Technologies. *Radial Basis Functions*. Published as a Chapter 3 in *Handbook of Neural Network Signal Processing*. Editors: Yu Hen Hu, Jenq-Neng Hwang. Boca Raton, Fla.: CRC Press, 2002.

7. W.Pedrycz. *Conditional Fuzzy Clustering in the Design of Radial Basis Function Neural Networks*. IEEE Transactions on Neural Networks, Vol. 9, No. 4, July 1998

8. T.Takagi, M.Sugeno. *Fuzzy Identification of Systems and Its Applications to Modeling and Control*. IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-15, No. 1, January-February 1985

9. M.Sugeno, T.Yasukawa. *A Fuzzy Logic Based Approach to Qualitative Modeling*. IEEE Transactions on Fuzzy Systems, Vol. 1, No. 1, February 1993

10. R.Babuska, L.Alic, M.S.Lourens, A.F.M.Verbraak, J.Bogaard. *Estimation of Respiratory Parameters via Fuzzy Clustering*. Artificial Intelligence in Medicine 21 (2001) 91-105 www.elsevier.com/locate/artmed

11. J.Moody, C.Darken. *Fast learning in networks of locally-tuned processing units*. Neural Computation, 1(2): 281 – 294, 1989

12. J.C.Bezdek *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum, 1981

13. T.P.Ryan. *Modern Regression Methods*. John Wiley & Sons, Inc., 1997

14. G.H.Dunteman. *Introduction to Linear Models*. SAGE publications, Inc., 1984

15. J.D.Tubbs. Regression Lecture Notes For *Stat 5313: Methods of Multivariate Analysis I*. Department of Mathematical Sciences, University of Arkansas. Spring Semester 2001. Chapter 2: *Matrix Approach to Linear Regression*. Retrieved from the Web on November 4, 2002: http://comp.uark.edu/~jtubbs/Courses/Regression/Notes/regression.mls.pdf

16. D.S.Broomhead, D.Lowe. *Multivariable functional interpolation and adaptive networks*. Complex Systems, 2: 321 – 355, 1988

17. M.J.D.Powell. *Radial basis functions for multivariable interpolation: a review*. In Algorithms for Approximation, 143 – 167, Clarendon Press, Oxford, 1987

18. L.Prechelt. *Early Stopping – But When?* Published in *Neural Networks: Tricks of the Trade.* Lecture notes in computer science; Vol. 1524. Edited by G.B.Orr, and K.R.Muller. Springer-Verlag Berlin Heidelberg 1998

19. Y.LeCun, L.Bottou, G.B.Orr, and K.R.Muller. *Efficient Back-Prop.* Published in *Neural Networks: Tricks of the Trade.* Lecture notes in computer science; Vol. 1524. Edited by G.B.Orr, and K.R.Muller. Springer-Verlag Berlin Heidelberg 1998

20. B.Kosko. *Neural Networks and Fuzzy Systems: a Dynamical Systems Approach to Machine Intelligence.* Prentice-Hall, Inc., 1992

21. D.E.Rumelhart, J.L.McClelland (eds.). *Parallel Distributed Processing*, vol. I. M.I.T. Press, Cambridge, MA, 1986

22. J.Moody. *Foreçasting the Economy with Neural Nets: A Survey of Challenges and Solutions.* Published in *Neural Networks: Tricks of the Trade.* Lecture notes in computer science; Vol. 1524. Edited by G.B.Orr, and K.R.Muller. Springer-Verlag Berlin Heidelberg 1998

23. J.Szymanski, W.Pedrycz, S.Frimpong. *An Artificial Neural Network Model For Prediction Of Truck Rear Suspension Pressures In Hauling Loaded Conditions.* IASTED Applied Simulation and Modelling Conference, Crete, Greece (June 2002), pp. 327 - 331